THE USE OF NUMERICAL METHODS AND MICROCOMPUTERS IN UNDERGRADUATE EXPERIMENTS

CPT Mark R. Stevens
HQDA, MILPERCEN (DAPC-OPA-E)
200 Stovall Street
Alexandria, VA 22332

Final Report, May 1988

UNCLASSIFIED/UNLIMITED

A Thesis submitted to
Rensselaer Polytechnic Institute, Troy, New York
in partial fulfillment of the requirements for the degree of
Master of Science

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

A196 2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT: UNCLASSIFIED/UNLIMITED
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: HQ DA, TAPA, DAIC-OPD; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code): 200 Stovall Street, Alexandria, VA 22332-0411	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)

The Use of Numerical Methods and Microcomputers in Undergraduate Experiments (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Stevens, Mark Richard

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM _____ TO _____	1988, May	135

16. SUPPLEMENTARY NOTATION
Thesis for Master's degree at Rensselaer Polytechnic Institute, Troy, New York

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The analytical power of a microcomputer in experiments is demonstrated through the use of numerical methods to solve the differential equations of motion. Both Heun's method (a second order Runge-Kutta method) and a fourth order Predictor-Corrector method are used to calculate approximate solutions to the equations of motion for physical systems. These rather tedious calculations are performed quickly with the computer and provide solutions for differential equations which are difficult or impossible to solve analytically in closed form. These methods are used to simulate and analyze the motion of a rotating disk and a physical pendulum, both including friction and air resistance. The motion of each system is recorded and plotted by the microcomputer. The Pascal software written allows comparison of the data with the numerical solution and will also calculate constants of the actual motion through a curve fitting iterative process. The ensuing laboratory setup of disk, pendulum and computer is then used in two college freshman experiments and one experiment at the level of a college junior. Keywords: Thesis (MS).

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: CPT Mark R. Stevens	22b. TELEPHONE (Include Area Code): (516) 271-2678 — 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

THE USE OF NUMERICAL METHODS AND MICROCOMPUTERS IN UNDERGRADUATE EXPERIMENTS

by

Mark R. Stevens

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Approved by the
Examining Committee:

Walter Eppenstein, Thesis Advisor

Philip Casabella, Member

Thomas Shannon, Member

Rensselaer Polytechnic Institute
Troy, New York

May 1988

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

First and foremost, I would like to acknowledge the extensive work done by Brian Davis in writing and debugging the 6809 Assembler codes used in the screen and graphics driving programs and the network communication routines. Secondly, I would like to thank my family for their patience and encouragement throughout my studies and research at RPI.

ABSTRACT

Advancements in the semiconductor industry in the last few years have led to more inexpensive and powerful microcomputers. It is inevitable that students will encounter some type of computer system at lower levels in the educational system. A vital part of a student's education should be to learn how to operate and utilize these systems. As the appearance of computers in educational institutions increases it is logical that they should be used in laboratory settings where their speed and analytical capabilities can be exploited for taking and analyzing data.

The analytical power of a microcomputer in experiments is demonstrated through the use of numerical methods to solve the differential equations of motion. Both Heun's method (a second order Runge-Kutta method) and a fourth order Predictor-Corrector method are used to calculate approximate solutions to the equations of motion for physical systems. These rather tedious calculations are performed quickly with the computer and provide solutions for differential equations which are difficult or impossible to solve analytically in closed form. These methods are used to simulate and analyze the motion of a rotating disk and a physical pendulum, both including friction and air resistance. The motion of each system is recorded and plotted by the microcomputer. The Pascal software written allows comparison of the data with the numerical solution and will also calculate constants of the actual motion through a curve fitting iterative process.

The ensuing laboratory setup of disk, pendulum and computer is then used in two college freshman experiments and one experiment at the level of a college junior. These exercises teach the power, speed and versatility of the computer in the laboratory; the power, usefulness and accuracy of numerical method solutions; and the characteristics of rotational and harmonic motion including the effects of friction and air resistance on each system.

1. INTRODUCTION AND HISTORICAL REVIEW

No field of science has advanced more rapidly in the past decade than the field of semiconductors and computers. The impact of smaller, faster, more efficient and affordable computing systems has been felt by the business and scientific community alike. This continuing trend has made the presence of personal computers (or microcomputers) in classrooms and laboratories no longer a luxury, but more of a necessity. With a great part of industry utilizing and depending on microcomputers it becomes extremely important that students currently in educational institutions become familiar with the use and possible applications of such systems.

Currently, most technical institutions do a good job of teaching their students the languages used on mainframe computers. Courses are offered in most higher level languages as well as the more specialized, lower level languages such as Assembler. There are even courses available which teach personal computing and the use of applications software. Unfortunately, an important area rich in possibilities is often overlooked. The speed, versatility and analytical capability of even the simplest microcomputers lend themselves to use in scientific laboratories. The introduction of such systems into the laboratory opens a wide variety of educational opportunities to students. These opportunities not only include analytical and computational possibilities but allow greater insight into physical processes through data collection and analysis.

There are a number of ways to utilize the capabilities of a microcomputer in the laboratory. Firstly, they can be used for data acquisition. Experiments can be conducted and controlled using continuously updated data from static sensors. Position and velocity calculations can be recorded by making repeated measurements in microseconds using an internal clock. Secondly, the analysis of data is greatly facilitated by a microcomputer. Curve fitting, differentiation, integration and simulations are all possible utilizing numerical techniques. Any combination of these capabilities in the laboratory

would be an important addition to a student's education.

Three experiments were developed for the purpose of teaching physics while demonstrating the power of microcomputers. Two of these are suitable for a freshman in college, and one is designed for the more advanced level of a junior. To demonstrate data collecting capability a physical pendulum and rotating disk were selected for investigation. Utilizing a notched wheel and photogates the microcomputer is used to take rapid measurements of the passing notches and record the resulting angular velocity of the motion as a function of time. Once the data is taken and recorded it is analyzed with the aid of the computer. Two numerical methods for solving differential equations are used to simulate the motion of each system. Heun's method is used for the freshman experiments and a fourth order Predictor-Corrector method for the junior experiment. Two computer systems are also needed. To take and analyze the data for the two freshman experiments a TRS - 80 Color Computer is used. A Zenith Z-140 personal computer is also employed because of a requirement for greater accuracy and speed for an iterative curve fitting routine in the junior's experiment.

The above selection of physical systems and numerical methods accomplishes a number of goals. First, it allows the student to more thoroughly investigate the physics involved with harmonic motion including frictional and viscous damping. The effects of varying different parameters are easily observed and verified with the experimental setup. Second, the usefulness of the computer is readily apparent in the ease in which the system records and plots the angular velocity of the motion. This aspect of the motion would be difficult to measure without the computer. Finally, using the numerical method demonstrates the power of the computer in calculating approximate solutions to normally unsolvable differential equations. Letting the student examine the accuracy of such methods builds confidence in the technique. After the completion of these experiments the student should feel comfortable using the computer and have a good

appreciation for the versatility and capabilities of the microcomputer in the laboratory. Armed with this experience the student should be more apt to utilize this powerful tool in his continued education and research.

2. THEORY

2.1 Physics Theory

Numerous physical systems are suitable for investigation using a computer. In choosing the physical pendulum and rotating disk two goals are met. First, both systems are relatively simple, easily constructed and understood. Second, in the case of the pendulum the differential equations of motion are not solvable analytically. Thus it is a good system for demonstrating the power and application of a numerical method. Also, because of the difficulty of solution, the physical pendulum is not often seen in the laboratory, and certainly not with large angle oscillations including frictional and viscous damping. While the solution to the rotating disk is less complicated, addition of viscous damping again demonstrates the numerical method as well as the concept of terminal velocity.

2.1.1 The Physical Pendulum

The simple pendulum all students learn about is actually an idealization of a physical pendulum. Any real pendulum is necessarily a physical pendulum. In developing the equations of motion we define d as the distance from the rotation axis to the center of mass, θ as the angle between the line from the pivot through the center of mass and the vertical and I is the moment of inertia of the oscillating body (Figure 2.1.1).

In the absence of any damping, using ω as the angular velocity, the equation of motion is

$$I\frac{d\omega}{dt} = -mgd\sin\theta$$

where m is the total mass and g is the acceleration due to gravity.[1] Solving this equation for ω and comparing it to the actual motion of a pendulum we can determine

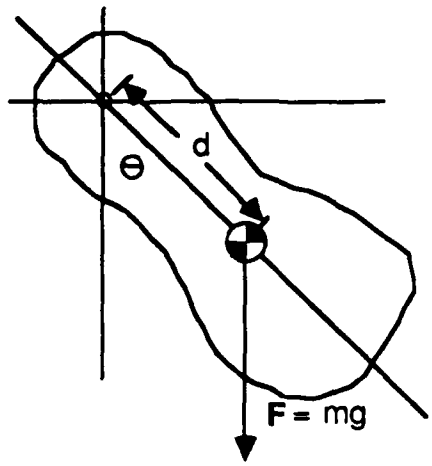


Figure 2.1.1. The Physical Pendulum

the system parameters such as the moment of inertia, providing damping can be neglected.

Unfortunately, friction and damping are inseparable parts of any real system. Naturally this affects the equations of motion. In the case of viscous damping the force can be approximated as being proportional to some power of the linear velocity of the resisting body moving through the viscous medium. In our case the major source of air resistance is due to plates fixed to the pendulum at a distance l. The torque resulting from this force is always opposed to the direction of motion and modifies the previous equation to

$$I\frac{d\omega}{dt} = -mgd\sin\theta \pm kl(\omega)^n$$

where n is an integer and k is a constant that accounts for the geometry of the plates as well as the proportion of the force to the velocity.[2] The plus or minus is only necessary for even values of n as the sign of ω will change this term correctly for odd n

values. In actuality the force is dependent on a polynomial which is a function of the velocity. However, since only one term dominates, and for the sake of simplicity, we will only use one term in the polynomial. The factor k will also vary depending on the state (temperature, pressure, humidity) of the viscous fluid that the system is in.

Finally, the pivot in the laboratory is made as close to ideal as possible by utilizing ball bearings but it also provides a frictional damping which must be considered. Because the friction involved in the bearing is rolling friction and the magnitude is small, its effect can be considered constant and opposite to the direction of motion. Denoting this constant opposing torque by f, the final equation of motion is

$$I\frac{d\omega}{dt} = -mgd\sin\theta \pm kl(\omega)^n \pm f$$

The exact solution of any of these equations is difficult, except when the small angle approximation ($\sin\theta = \theta$) is made for the undamped case. An accurate approximate solution to these equations can be obtained though, through the application of numerical methods.

2.1.2 The Accelerated Disk

Generally students learn about terminal velocity by considering a sky diver or an object dropped from a tall building. By using a rotating disk it is easy to demonstrate this concept in the laboratory. In developing the equation of motion for this system r is defined as the radius from the axis of rotation to the point where the accelerating weight is attached, ω is the angular velocity and I is again the moment of inertia of the disk (Figure 2.1.2). Using simple Newtonian mechanics we know that mg - T = ma where T is the tension in the cord and a is the downward acceleration of the mass. We also know that the torque caused by the weight is given by $Tr = I\frac{d\omega}{dt}$. Noting that $a = r\frac{d\omega}{dt}$ we can eliminate the acceleration between the two equations and solve for the tension T. Thus, neglecting damping, the total torque acting on the disk can be written[3]

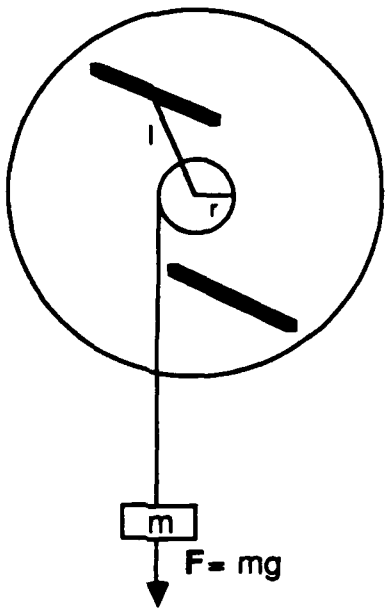


Figure 2.1.2. The Accelerated Disk

$$I\frac{d\omega}{dt} = \frac{mgrI}{I+mr^2}$$

In order to attain a terminal velocity the effects of air resistance must be included. Again the viscous damping force is proportional to some power of the linear velocity of the plates on the disk which cause most of the resistance. This modifies the total torque to be

$$I\frac{d\omega}{dt} = \frac{mgrI}{I+mr^2} - kl(k\omega)^n$$

where l is the distance to the center of the resisting plates and k is the proportionality constant. This equation of motion would demonstrate terminal velocity, however we have neglected the rolling resistance of the bearings. Including this effect we obtain

$$I\frac{d\omega}{dt} = \frac{mgrI}{I+mr^2} - kl(\omega)^n - f$$

where f again represents this constant factor. Again, terminal velocity is achieved when the net torque is zero.

As was the case with the simple pendulum, these equations of motion considering the viscous forces are not easy to solve analytically. We again turn to numerical methods to provide accurate approximate solutions to this motion.

2.2 Mathematical Theory

Numerical methods exist for a variety of mathematical tasks. We are concerned with only those methods useful in solving second order differential equations. Different methods have advantages and disadvantages involving speed, accuracy and application. Numerical techniques for solving differential equations are divided into one step and multistep methods. Both types of methods are utilized in the simulation routines for the experiments. The primary one step technique is the Runge-Kutta method while the Predictor-Corrector method is representative of the multistep techniques.

2.2.1 Runge-Kutta Methods

Runge-Kutta methods are designated as one step methods because they only require the initial conditions in order to calculate a solution. Because of this capability they are also classified as being self starting. They approximate the value for the next point of the solution to a differential equation given an incremental step size for the independent variable using only the value of the point before. The general one step method is an iteration that takes the form [4]

$$y_{i+1} = y_i + h\Phi(x_i,y_i;h) \quad i = 0,1,..,n-1$$

The function $\Phi(x_i,y_i;h)$ is called the increment function and tells us how to proceed from

an estimated y_i for $y(x_i)$ to the next point y_{i+1} for $y(x_{i+1})$. The variable h is the step size taken with the independent variable. The definition of the function $\Phi(x_i,y_i;h)$ depends on the particular method.

If we think of differential equations as specifying the slope of the solution at points in space we can construct flow fields which give us an idea of the form of the solution. For example, the differential equation $y' = y(2 - y)$ has the flow field shown in Figure 2.2.1.[5] As we move from the point x_i to x_{i+1} the nature of the flow field governs the change in the solution y(x) between the two points. It makes sense to sample the flow field between the points and then specify the increment function as a weighted average of the samplings. Thus

$$\Phi_k(x_i,y_i;h) = A_1 f(\theta_1,\gamma_1) + A_2 f(\theta_2,\gamma_2) + \ldots + A_k f(\theta_k,\gamma_k)$$

The problem reduces to choosing the weights A_j and the specific sample points (θ_j,γ_j) in the interval.

Before specifying how the Runge-Kutta method selects these weights, we need to digress to the more familiar Taylor series method. If we expand our function in a Taylor series we have

$$y(x_1) = y(x_0) + y'(x_0)h + y''(x_0)\frac{h^2}{2!} + y^k(x_0)\frac{h^k}{k!} + y^{k+1}(\theta)\frac{h^{k+1}}{(k+1)!}$$

When y(x) is sufficiently differentiable the more terms carried, the more accurate the approximation. Since we are given $y'(x) = f(x,y)$ (our equation to solve) we can calculate the higer order derivatives as necessary. Our increment function is then[6]

$$\Phi_k \equiv T_k(x,y;h) = f(x,y) + f'(x,y)\frac{h}{2} + \ldots + f^{k-1}(x,y)\frac{h^{k-1}}{k!}$$

The difficulty with this method is that it requires the knowledge of the higher order derivatives. As a result, higher order Taylor methods are rarely used.

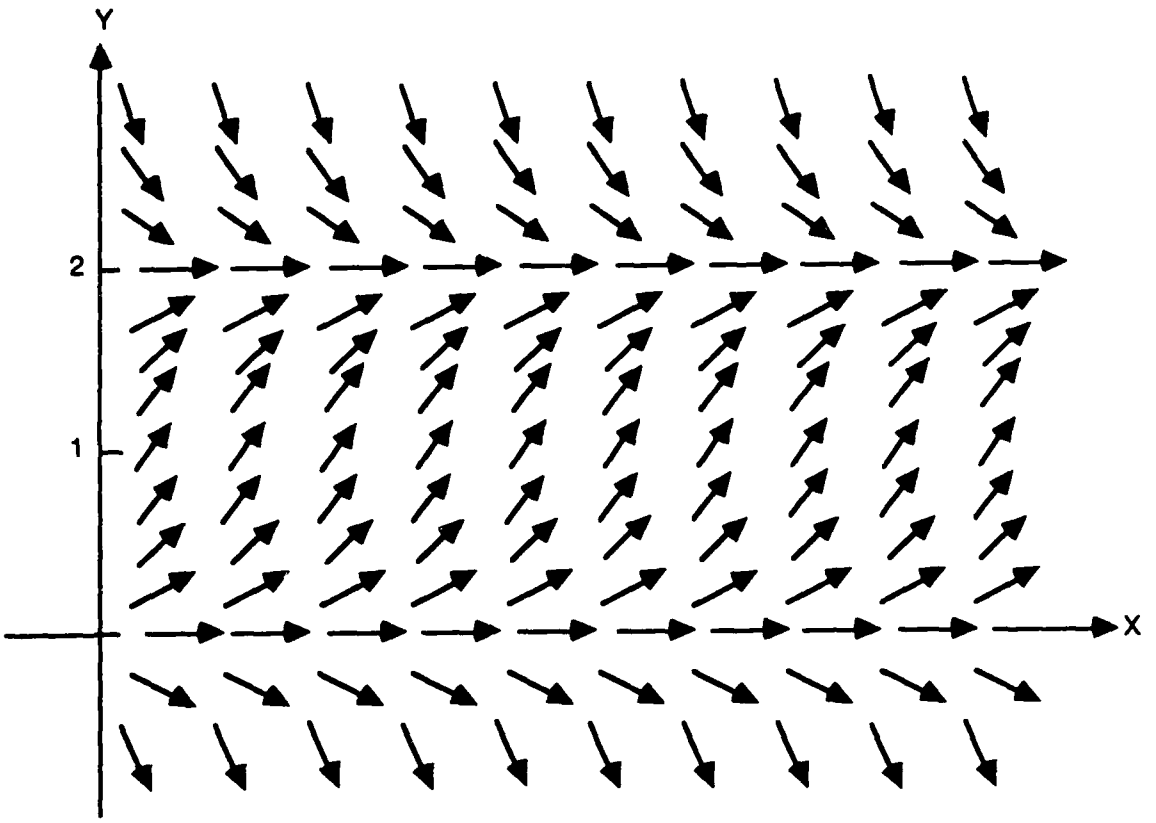


Figure 2.2.1. Example Flow Field

The German mathematicians C. Runge and M. W. Kutta sought to circumvent the problem of higher order derivatives inherent in the Taylor method. The basis for the Runge-Kutta method is to choose the sampling points and weighting functions A_j so that they agree as well as possible with the increment function for the equivalent Taylor series method. Note that the familiar Euler's method $y_{i+1} = y_i + hf(x_i,y_i)$ is both a first order Taylor series and Runge-Kutta method. The simplest way to demonstrate the method is to use an example.

For a second order Runge-Kutta method we need two sample points. Choosing the points (x_i,y_i) and $(x_i+\beta h, y_i+\beta hf(x_i,y_i))$, where β is undetermined, the increment function is of the form

$$\Phi_2(x_i,y_i;h) = A_1 f(x_i,y_i) + A_2 f(x_i+\beta h, y_i+\beta h f(x_i,y_i))$$

We want to match this with the Taylor series increment function

$$T_2(x_i,y_i;h) = f(x_i,y_i) + \frac{h}{2}[f_x(x_i,y_i) + f_y(x_i,y_i)f(x_i,y_i)]$$

To do this we expand the function associated with A_2 in a Taylor series yielding

$$f(x_i+\beta h, y_i+\beta h f(x_i,y_i)) = f(x_i,y_i) + f_x(x_i,y_i)\beta h + f_y(x_i,y_i)\beta h f(x_i,y_i) + E$$

where the remainder E is of the form Ch^2. Substituting into Φ_2 and collecting like terms we find '

$$\Phi_2(x_i,y_i;h) = (A_1 + A_2)f(x_i,y_i) + A_2 h[\beta f_x(x_i,y_i) + \beta f_y(x_i,y_i)f(x_i,y_i) + Ch]$$

Comparing this with $T_2(x_i,y_i;h)$ we get a fairly close match if $A_1 + A_2 = 1$ and $A_2\beta = \frac{1}{2}$. This means that

$$\Phi_2(x_i,y_i;h) = T_2(x_i,y_i;h) + A_2 Ch^2$$

so the error in approximating the function is partly due to the error in the Taylor expansion $T_2(x_i,y_i;h)$ of the order Bh^2 and to the error in matching $T_2(x_i,y_i;h)$ of the order A_2Ch^2. Looking at the relationship of the coefficients we see that $A_2 = \frac{1}{2\beta}$ and $A_1 = 1 - \frac{1}{2\beta}$. There are an infinite number of solutions for these parameters, however a natural choice is $\beta = \frac{1}{2}$ yielding

$$y_{i+1} = y_i + hf(x_i+\frac{h}{2}, y_i+\frac{h}{2}f(x_i,y_i))$$

This is the modified Euler's method where the sampling point is one half of the way into the interval. Another choice is $\beta = 1$ which yields

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i,y_i) + f(x_i+h,\ y_i+hf(x_i,y_i))]$$

This is Heun's method and it samples the flow field at the beginning and the end of the interval and gives each value an equal weight. Figure 2.2.2 is a graphic example of how the sampling method works for Heun's method. Both of these methods are of second order and have the same magnitude of error, proportional to h^2. Heun's method will be used extensively.

It is possible to use this technique to develop higher order methods, the benefit being greater accuracy, the cost being increased calculation time. A popular fourth order method which will also be employed is[8]

$$y_{i+1} = y_i + \frac{h}{6}[K_1 + 2K_2 + 2K_3 + K_4]$$

where

$$K_1 = f(x_i,y_i)$$
$$K_2 = f(x_i+\frac{h}{2},\ y_i+\frac{h}{2}K_1)$$
$$K_3 = f(x_i+\frac{h}{2},\ y_i+\frac{h}{2}K_2)$$
$$K_4 = f(x_i+h,\ y_i+hK_3)$$

Note that this method samples the midpoint of the interval twice and five calculations are required to step to the next point where the whole process is repeated. Accuracy can also be improved by taking smaller steps. In fact, it is possible to vary the step size h based on the behavior of the solution to maintain the error of the approximation below some established limit. Such variable step techniques as well as convergence and error analysis are thoroughly discussed in the references.[9]

The algorithm used for the fixed step 4th order Runge-Kutta method is as follows.

1. Calculate the slope values at the starting point (K_1).

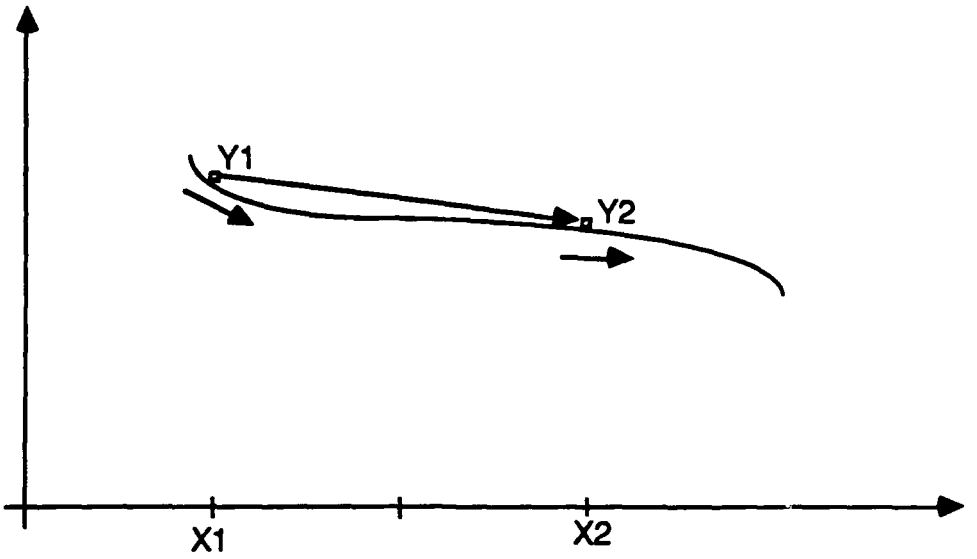


Figure 2.2.2. Runge-Kutta Sampling Method

2. Use the estimated slope to calculate the solution's value at one half the interval.

3. Use the estimated value to find the slope at one half the interval (K_2).

4. Use this slope to again calculate the solution at one half the interval.

5. Use the estimated value to find the slope at one half the interval (K_3).

6. Use this slope to calculate the solution value at the end of the interval.

7. Use the estimated value to find the slope at the end of the interval (K_4).

8. Calculate the solution's value at the end of the interval using the weighted average of the slope values K_1 to K_4.

9. Use the final point as the new initial point and return to step 1.

2.2.2 Predictor-Corrector Methods

The one step methods descussed above only require one point to calculate the next point after sampling the flow field in the interval. This necessarily requires a great number of calculations for each point (five for the fourth order method above). There is

a class of numerical procedures called linear multistep methods which use "back values" y_n, y_{n+1}, . . , y_{n+k-1} to determine y_{n+k}. These methods, one of which is the Predictor-Corrector, use the past history, or trends, of the numerical solution at equally spaced points x_n, x_{n+1}, . . , x_{n+k-1} to estimate the solution at $x = x_{n+k}$. The general form for a multistep method is [10]

$$y_{n+k} = -\sum_{j=1}^{k} \alpha_j y_{n+j} + h \sum_{j=1}^{k} \beta_j f(x_{n+j}, y_{n+j})$$

where again $y' = f(x,y)$. It is called a linear k - step method because it is linear in y_{n+j} and $f(x_{n+j}, y_{n+j})$ and it requires k back values to calculate y_{n+k}. Normally we demand that $\alpha_k = 1$ and that $|\alpha_0| + |\beta_0| > 1$. If $\beta_k = 0$ then the method is explicit. If $\beta_k \neq 0$ then the value of y_{n+k} is needed to evaluate the right side of the equation to solve for y_{n+k} and the method is implicit. The greatest advantage of this type of method is that we can save the past evaluations of $f(x_{n+j}, y_{n+j})$ and then only need to make several evaluations of $f(x,y)$ to obtain y_{n+k}. This reduces the calculations required below that of comparable Runge-Kutta methods speeding calculation times. These methods are not self starting because they require a certain number of back values before they can proceed. A one step method is generally employed to start a multistep method.

In practice, the implicit and explicit methods described above are used in pairs. The explicit method is used to predict the solution's value at a point and this value is then used in the implicit method to correct the initial solution value. This is the origin of the name Predictor-Corrector. The most popular methods are Adams methods. These are quadrature methods based on aproximations to the integral of the form [11]

$$\int_{x_{n+k-1}}^{x_{n+k}} f(x)dx \simeq h[A_0 f(x_n) + A_1 f(x_{n+1}) + \ldots + A_k f(x_{n+k})]$$

Graphically the method is demonstrated in Figure 2.2.3.

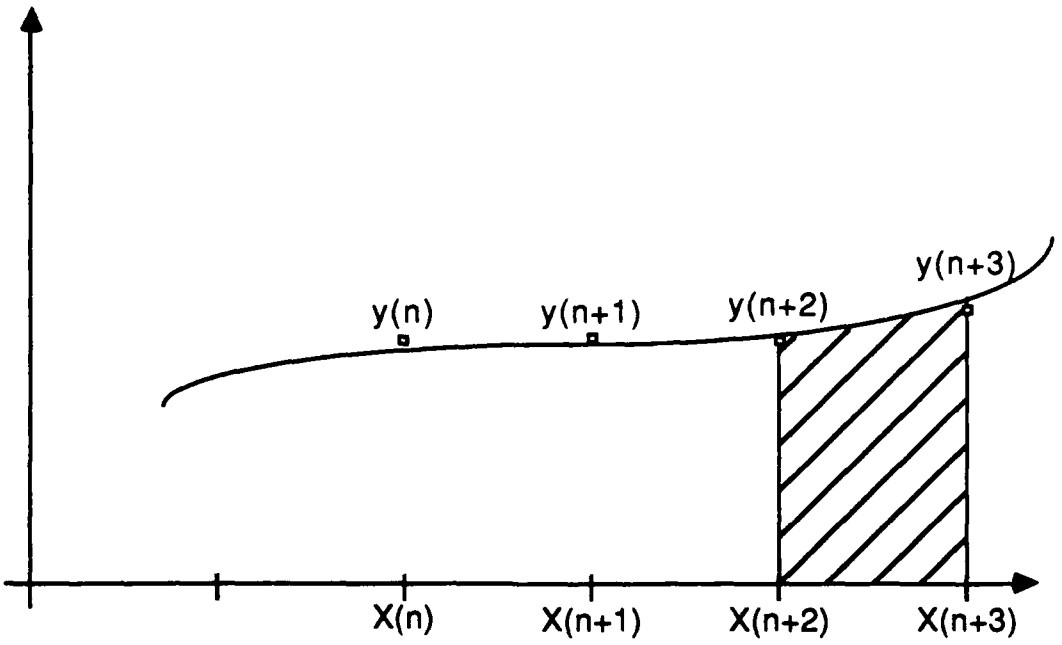


Figure 2.2.3. Quadrature Methods

An explicit Adams-Bashforth method looks like

$$y_{n+k} - y_{n+k-1} = h[A_0 f_n + A_1 f_{n+1} + \ldots + A_{k-1} f_{n+k-1}]$$

and an implicit Adams-Moulton method has the form

$$y_{n+k} - y_{n+k-1} = h[A_0 f_n + A_1 f_{n+1} + \ldots + A_k f_{n+k}]$$

where $f_m = f(x_m, y_m) = y'_m$. It turns out that the error of a k step explicit method is of the same order as a k-1 step implicit method. If the error is of the same order it reduces the number of iterations needed for the corrector to produce an accurate value limiting the number of calculations. For that reason the Predictor-Corrector pairs usually differ by one step and the corrector is normally applied once, although it can be iterated several times.

Using the fourth order Runge-Kutta method described earlier to start the multistep method it seems logical to maintain the same level of accuracy with a fourth order Predictor-Corrector method. Using interpolatory numerical integration techniques the A_js for different order methods can be specified and standard values are given in the references. Identifying $\bar{y}$ as the predicted solution value to be used in the implicit corrector we can combine the three step Adams-Bashforth predictor

$$\bar{y}_{n+3} = y_{n+2} + \frac{h}{12}[23f_{n+2} - 16f_{n+1} + 5f_n]$$

with the two step Adams-Moulton corrector

$$y_{n+3} = y_{n+2} + \frac{h}{12}[5f(x_{n+3},\bar{y}_{n+3}) + 8f_{n+2} - f_{n+1}]$$

to obtain a fourth order method requiring three back points and only two evaluations. Note that in comparison the same order Runge-Kutta method requires five calculations to obtain the same point. This indicates that the Predictor-Corrector method should be significantly faster. This method can also be used in variable step techniques and the resulting error monitored.[12] Variable step techniques are not really necessary, however, if the solutions to the differential equation are generally well behaved.

2.2.3 Extension to Higher Order Differential Equations

Up to this point the methods discussed only addressed first order differential equations. The equations of motion we are concerned with are second order differential equations. To apply the preceding methods to higher order equations we first convert the nth order equation into an equivalent system of n first order equations. Thus $u_1(x) = y(x)$, $u_2(x) = y'(x)$, . . , $u_n(x) = y^{n-1}(x)$. Then using vector notation we set

$$\mathbf{u} = \begin{vmatrix} u_0(x) \\ u_1(x) \\ . \\ . \\ u_n(x) \end{vmatrix} \qquad \mathbf{F}(x,\mathbf{u}(x)) = \begin{vmatrix} u_1(x) \\ u_2(x) \\ . \\ . \\ f(u_1(x),u_2(x),..,u_n(x)) \end{vmatrix}$$

This gives an equivalent first order system of equations in vector form where

$$\mathbf{u}'(x) = \mathbf{F}(x,\mathbf{u}(x))$$

and $\mathbf{u}(x_0) = \mathbf{u}_0$ are the initial conditions.[13] Now we just apply the numerical methods to the vector equation and calculate vector quantities. As an example, Euler's method becomes

$$\mathbf{u}_{i+1} = \mathbf{u}_i + h\mathbf{F}(x_i, \mathbf{u}_i)$$

and one step methods take the form

$$\mathbf{u}_{i+1} = \mathbf{u}_i + h\Phi(x_i, \mathbf{u}_i;h)$$

This automatically produces the value of the function and any derivative up to the order of the original equation at any given point. In the physical systems considered previously this vector technique gives us the value of the angular displacement and angular velocity automatically at each point.

The algorithm for the fourth order Predictor-Corrector method is as follows.

1. Calculate the first three back points with a one step method.
2. Calculate the initial functional values at each of the three back points (f_1,f_2,f_3).
3. Apply the predictor and estimate $\bar{y}_{n+3}$.
4. Calculate the function's value f_{n+3} at the estimated point $\bar{y}_{n+3}$.
5. Apply the corrector to obtain a more accurate y_{n+3}.

6. Move the functional values up and calculate the new value f_3 at the corrected point y_{n+3}.

7. Return to step 3 (functional values are stored in memory and moved up for each pass).

3. EXPERIMENTAL APPARATUS AND DATA ACQUISITON

3.1 Experimental Setup

Both the physical pendulum and rotating disk use the modified TRS-80 Color Computer system from Radio Shack (Appendix A). For the junior's experiment a Zenith Z-140 PC with 8087 math coprocesser was also used. The heart of this experimental apparatus is the hardware used to measure the angular velocity of either system. This setup can actually measure the angular velocity of any physical configuration you wish to attach to the slotted disk (Figure 3.1). The motion is recorded through the use of a ten centimeter diameter slotted aluminum disk mounted on a ball bearing hub. There are ninety slots equally spaced around the circumference of the disk. Straddling the slotted area of the disk are two photogates which are capable of signalling as each slot passes. The light beam is broken as the solid portion of the disk between the slots passes when the disk rotates. Since we know the angular position of each notch the angular velocity of the disk is easily calcualated by counting the number of notches that pass in one tenth of a second.

In order to convert the signal from the photogates into a usable (countable) signal for the computer, a hardware circuit is used to convert the signal and provide the input to the cartridge slot of the TRS-80 (Figure 3.2). This circuit actually counts edges and not notches. Thus each photogate will register two counts (two edges) for every notch. Considering the redundant photogates there are actually four counts per notch. This duplicity must be taken into account when calculating the velocity.

Attached to the slotted disk is another thirty centimeter diameter aluminum disk. For the terminal velocity experiment all that is required is the addition of two adjustable plates for air resistance. Two rectangular plates measuring 21 centimeters by 17 centimeters were mounted 22 centimeters from the center axis for this purpose. To construct a physical pendulum all that is needed is a weight to unbalance the disk.

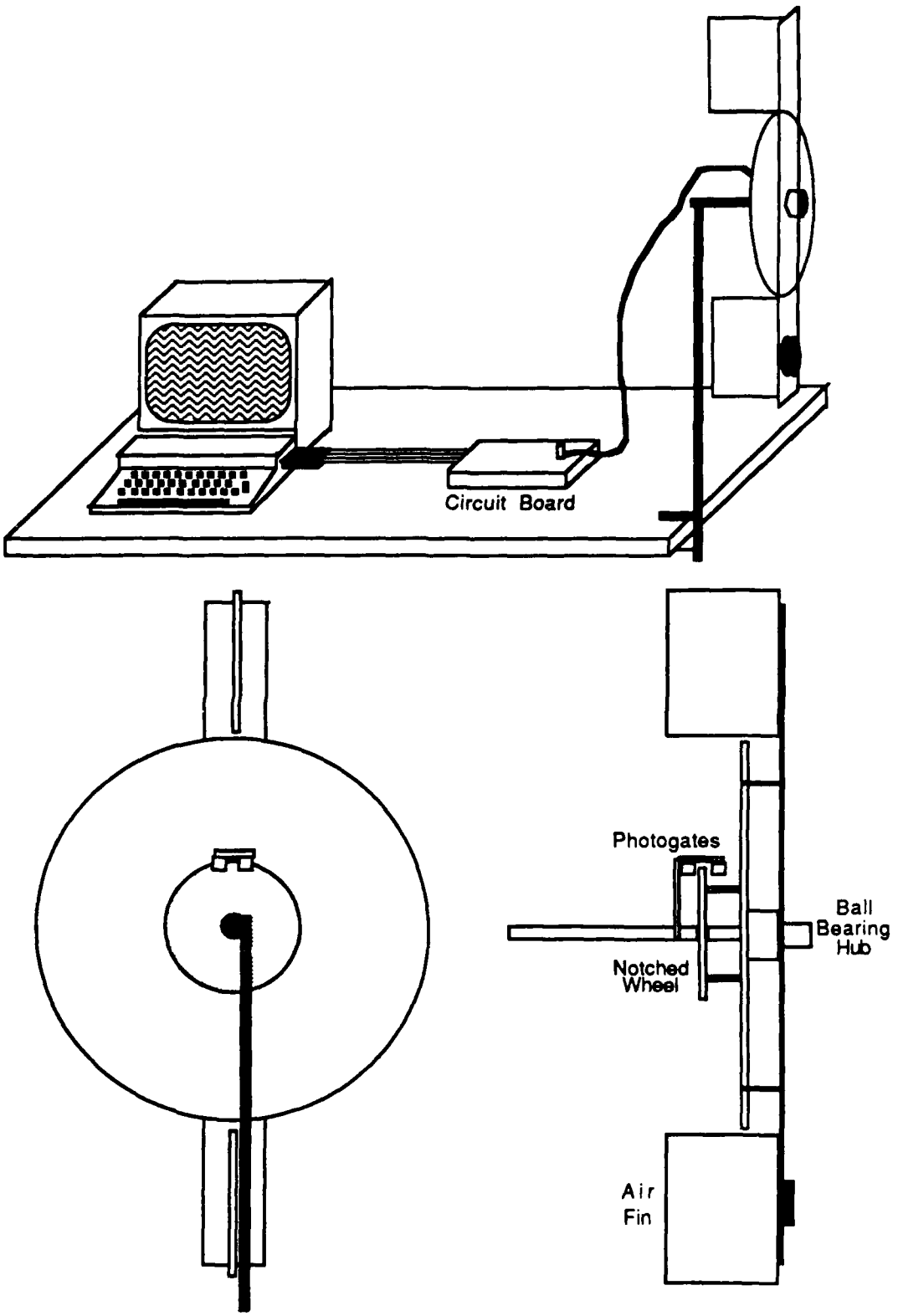
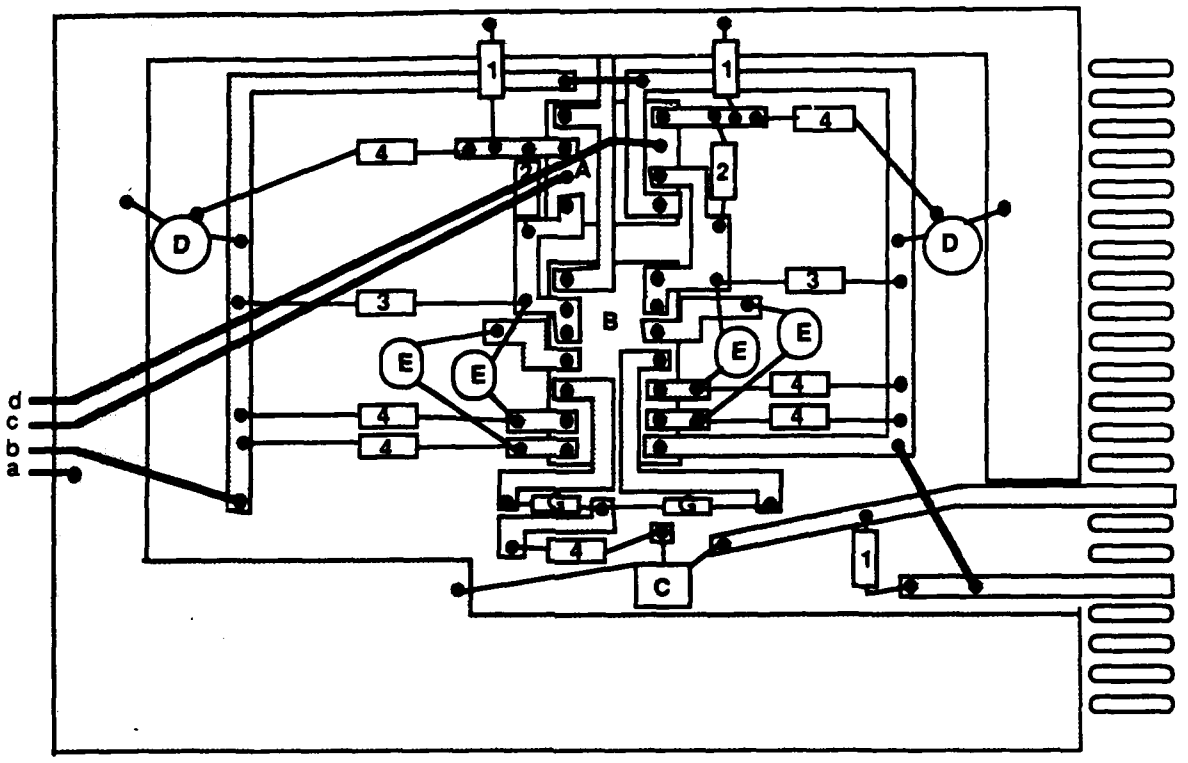


Figure 3.1. Experimental Setup



KEY

A - M8428 LM 393N	1 - 1000 Ohm
B - SCL 4011AE 7427	2 - 680000 Ohm
C - 225 2N 3904	3 - 4700 Ohm
D - 2000 Ohm Variable	4 - 10000 Ohm
E - .0242mf 50V	5 - 240 Ohm
F - OP 8825 8203	6 - 83 Ohm
G - 1N	7 - 100 Ohm

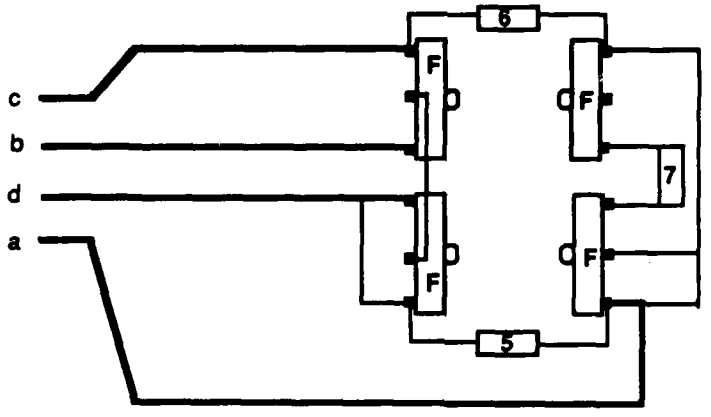


Figure 3.2. Counting Circuit

To accomplish this a 200 gram mass is attached at the same location as one of the air resisting plates. This setup is very flexible and other configurations are possible using the slotted disk as a foundation.

3.2 Software

The hardware described above provides a physical system and a means of measurement but the photogate signals must be converted into meaningful data and analyzed by programmed routines. There are several sets of programs listed in Appendix B. Two different programing languages were used. 6809 Assembler was used in the procedures for taking the data and plotting as well as the screen driver routines. These routines can then be linked to the interactive Pascal main program by using DEFT Pascal Workbench. [14] Both of the freshman experiments use this language and the TRS-80 system exclusively. For the junior level experiment only the data acquisition and conversion uses the TRS-80 and DEFT Pascal. The data must then be input into the Zenith Z-140 PC where the analysis routine used is written in Turbo Pascal. [15]

All of the routines use the same technique to record the data. The signal from the photogates automatically increments a 6809 register by one for each pulse. Every tenth of a second the contents of the register is recorded in a vector of data points and zeroed to receive the next set of signals from the photogates. This data vector is then used by each routine. Only the Jpendata routine actually converts the counts directly into angular velocities for input into the Zenith microcomputer.

The structure of both of the routines for the rotating disk (Airwheel) and the physical pendulum (Freshpen) are essentially the same. Initially the student chooses to take data or run a simulation of the motion. If he chooses to take data the subroutine instructs the student how to simultaneously release the system (pendulum or disk) and trigger the counting routine. The data can then be plotted for viewing. After viewing the data the student may continue or terminate the program. Continuing, the program

returns to the starting point where the data can be taken again, if necessary, or analyzed with the help of the simulation. Once in the simulation subroutine the physical parameters of the system are entered and the treatment of the air resistance selected. One can choose to neglect air resistance or include it linearly or quadratically in the differential equations. The step size in time for use in Heun's method must be selected next. Heun's method is used to calculate the simulated motion which is plotted and compared to the actual data points. The previous simulation calculated is also plotted so that trends can be indentified. The program then allows changes in the step size, treatment of the air resistance and the initial conditions, in that order. If no changes are desired the student may elect to terminate the program or return to the beginning to repeat the simulation or take new data.

The Physpend program on the Zenith PC is a little more powerful and sophisticated than the above two routines in keeping with the Zenith's capabilities. When the program is initiated there are five options available to the student. First, selecting Takedata permits entering the data points provided by the Jpendata program. After the data is entered it may be plotted and also shifted if necessary. The routine then loops back to the beginning where the student can select the simulation option. Here, as above, the type of configuration is selected, the physical parameters of the system entered, and the time step specified. Now, however, due to the greater computing power of the Zenith, the fourth order Predictor-Corrector method is used to provide a more accurate approximate solution. The simulation may then be plotted. If a simulation has already been done then a second simulation can be calculated and plotted. If data has been taken and simulations completed the data can be compared graphically with each simulation and each simulation compared to the other. A printed version of the plot can be obtained using the print screen key command. Examples of such output are in Appendix C. The Smallangle option uses the small angle

approximation and stores the resulting calculated motion as a first simulation so that the accuracy of this approximation can be investigated. Finally, with the Compare option the computer will calculate either the moment of inertia, coefficient of friction or the coefficient of air resistance for the physical pendulum. This routine uses an iterative technique and calculates the simulated motion and compares it with the data file point by point. If the value is outside the tolerance set by the student, it will abort the calculation, increment the variable and recalculate the motion until all data points are within the tolerance or until an upper limit, also set by the student, is reached. Here the accuracy of the numerical method is very important. This was the primary reason for selecting a fourth order Predictor-Corrector method. The final value for the selected variable is output and the procedure returns to the options menu at the beginning of the program. At any time when returned to this menu the program can be terminated by selecting the Quit option. The flow charts and program listings in Appendix B provide further details.

4. DISCUSSION

4.1 Goals and System Selection

Several goals were established in designing the experiments which incorporate the preceding theory and equipment. First and foremost, the experiments need to teach a meaningful concept in physics. Secondly, with the current power and availability of microcomputers the student should be taught to appreciate the value of such systems in the laboratory, whether in taking or analyzing data. Finally, the student should be introduced to methods which are available for problems without analytical solutions. These three experiments accomplish all of these goals.

The selection of the physical system investigated is of obvious importance in attaining all of the above goals. The choice must lend itself to computer data acquisition and analysis, as well as the application of a numerical method, while involving more than trivial physics. Obviously there are numerous possibilities. The area of mechanics was selected due to the sparsity of existing experiments on mechanical systems. Also, as equations of motion are in differential form, these systems lend themselves to numerical analysis. In choosing the specific experiment an endeavor was made to find systems which perhaps have been addressed in courses yet are seldom seen in demonstrations or laboratories.

Every student learns about terminal velocity in his introductory physics course. Perhaps he may even solve a representative problem concerning a falling sky diver, but sky diving is difficult to demonstrate in the laboratory. The rotating disk with air resistant fins is a compact system which easily demonstrates this concept. It also reinforces the concepts of rotational motion and the moment of inertia of a rigid body. Measuring the angular velocity of such a system by hand would be unreasonably difficult, so the ease of recording this motion with the slotted disk and TRS - 80 provides a good demonstration of a microcomputer's power and usefulness. Finally, the

equations of motion for this system are not analytically solvable when viscous forces are taken into account. Thus the power and application of a numerical method is easily demonstrated. The rotating disk in a viscous retarding fluid is a suitable choice considering the desired goals.

No one can deny the importance of harmonic motion in modern physics. Often the concept is introduced with the aid of a spring-mass system or simple pendulum. Unfortunately, pendulums in real life are actually physical pendulums and their oscillations are not always small. The physical pendulum and large angle oscillations are often avoided, or treated superficially, due to their more complicated nature and difficult solution. It is very appropriate, however, for our purposes. This system helps present the ideas of harmonic motion and the effects of external forces. Over, under and critical damping are easily ivestigated as well as the real world effects of friction in the bearings of the apparatus. The accuracy and application of the standard small angle approximation can be investigated and the concepts of rotational motion and the moment of inertia reinforced. The computer's ability to easily record this motion again demonstrates its value in the laboratory. Numerical methods are definitely applicable to this system because, even neglecting damping forces, an analytical solution to the equations of motion is not possible without the small angle approximation. Thus the power and application of a numerical method is easily shown. In view of the desired goals the physical pendulum is also a suitable, and possibly more flexible system to investigate.

4.2 Experimental Design

Using the above goals and physical systems three experiments were developed. Two of these are intended for college freshmen and are done exclusively with the TRS - 80 computer. The third experiment is intended for college juniors in physics and thus requires greater knowledge and initiative. It also requires a Zenith Z-140 PC computer

or comparable IBM compatible system. The instructions for all three experiments reflect the different levels of expertise in their presentation and detail (Appendix D).

The structure of both of the freshman experiments are similar. First, the laboratory setup and general concepts are introduced. Because freshmen are not as familiar with the theory involved the second section reviews the necessary physics and the equations of motion for each system. The instructions for the execution of the experiment are then given, including specific questions that must be answered. The moment of inertia is calculated for each system using standard shapes and the parallel axis theorem. The initial parameters of the setup and initial conditions are measured and recorded. To make the operation of the computer more transparent the terminal velocity experiment requires the student to calculate a few of the data points by hand using the improved Euler's method. This is a good exercise to help the student recognize the power of the computer and understand how the numerical method is applied. Next the actual motion of the disk or pendulum is recorded. The student then uses the computer and the programed numerical method to create a plot of the expected theoretical motion. He then compares the theoretical and actual motion in order to draw conclusions about the effects of viscous forces and the values of certain constants. The configuration of the system can then be changed, the new motion recorded, compared to theory, and similar conclusions can be drawn. Finally, the accuracy of the numerical method and its dependence on the step size is investigated pointing out the possible limitations and concerns of numerical methods.

The experiment intended for juniors is concerned with the physical pendulum only. It is much less structured than the two freshman experiments. The instructions only cover the necessary information to operate the programs on both of the computers. Some areas of interest are suggested, however the scope and depth of the investigation is left up to the student's own initiative. The student can easily investigate a large

number of things because of the greater versatility and power of the Zenith computer. Areas include, but are not limited to, small and large angle oscillations, the small angle approximation, the coefficient of rolling friction in the ball bearings, all categories of viscous damping, the accuracy of the calculated moment of inertia compared to the computer's value obtained through curve comparison, and so forth. The accuracy of the numerical method with varying time step size can also be investigated in a similar manner to the freshman experiments. The lack of structure and dependence on student initiative is designed to help develop the student's natural curiosity, good experimental methods and techniques.

In the fall of 1987 the two freshman experiments were given to selected college physics majors as part of their normal junior laboratory course. The juniors were instructed to rigorously tax the software and experimental setup as a means of trouble shooting and debugging. As a result of their thorough investigation minor changes in software were affected, however the entire system as a whole ran smoothly. More importantly, the objectives set for these experiments seemed to be genuinely met. Most students indicated that the exercises were valuable in conveying the concepts of terminal velocity and harmonic motion. They also felt the experiments reinforced the usefulness of the computer in the laboratory and demonstrated an application of a pertinent numerical method. The junior's experiment was not tested in this manner, but when the freshman experiments were given to the juniors they were not provided the instructions in Appendix D. Instead their instructions were less structured, much like those intended for the juniors. The performance of the juniors on these experiments indicates that the design and structure of the junior's experiment should also successfully meet the stated goals.

4.3 Limitations and Possible Improvements

As is normally the case, these experiments have some limitations and are subject to improvements. While large angle oscillations can be accommodated for the pendulum, excessively large angles (greater than 120 degrees) will cause plots to go off the screen for displacement graphs. Likewise, if the angular velocity is large the resulting graphs will also go off the screen. This can be handled by properly selecting the system, or by changing the plotting scale in the program if necessary. Thus, the setup will accomodate any configuration of pendulum and disk, subject to some common sense.

There are also several areas where errors can be introduced. First, because the notched disk has a fixed number of notches, the values recorded for the angular velocity form a discrete set. If the actual value for the angular velocity is between two of these values it will be rounded down to the lower value. Accuracy is less important for the freshman experiments as comparisons are graphical and the resolution is less necessary. For the junior's experiment the accuracy of the data is much more important since numerical calculations are conducted with the actual values for the angular velocity. A suggested improvement would be to increase the number of notches on the disk. Doubling the number of notches would cut the inaccuracy in taking the data in half. The angular velocity values would still form a discrete set, but there would be twice the number of values.

Another source of inaccuracy and a potential problem is the initiation of the counting routine and the simultaneous release of either the disk or pendulum. Since all of the simulations assume that the initial angular velocity is zero, any motion before the computer begins to take data will make it impossible to match the data curve. For the two freshman experiments there is no recourse but to retake the data. The program for the Zenith PC does have a routine that will shift the data to correct for initiation

problems. Remedies for this difficulty take two forms. First, it should be possible to utilize some sort of electronically controlled mechanism to release the pendulum or disk at the same time the counting routine is initiated from the keyboard. This would reduce the error significantly. Another option would be to use the data itself to provide an initial value for the angular velocity. This would only work if the system was already moving when the computer began to take data. If it was not in motion the resulting accuracy would be no better than if the system had been released by hand.

Finally, it seems to be counter productive to take data with one microcomputer and then input numerical values by hand into another. The Zenith PC is capable of accepting input through its RS 232 serial port. Although the TRS - 80 uses an 8 bit processor as opposed to the Zenith's 16 bit processor it would be possible for the TRS - 80 to transmit its data to the Zenith. This was explored, however modifications to the peripheral TRS - 80 where the data was taken seemed to preclude any transmission back out of the system. It should also be possible to take data directly with the Zenith through the RS 232 serial port or an expansion slot. This was not explored as thoroughly because of the existing data taking capabilities of the TRS - 80. Certainly this is the way to proceed in the future considering the limited speed and memory of the TRS - 80 compared to the Zenith PC.

5. CONCLUSIONS

With the institutional desire to provide a complete and practical education it has become important to expose students to the power and versatility of microcomputers in the laboratory. This is becoming increasingly important as these systems are more readily available in universities and industry. The incorporation of the physcical pendulum, rotating disk and a numerical method for solving differential equations into the laboratory using a microcomputer helps accomplish these goals. The computer is used as an educational tool to teach the applicable physics of each system. Using the same computer to collect and analyze the data demonstrates its power and versatility. Finally, the use of a numerical technique demonstrates a powerful and useful method for solving analytically difficult problems.

There are other possible structures for the experiments presented which could accomplish the same results. The limited exposure to current students seems to indicate, however, that the proposed experiments will be successful in fulfilling the purpose for which they were designed. As needs change or capabilities in the laboratory increase the systems and software are easily modified to accommodate different physical configurations or changed emphasis for each experiment. A great number of possibilities exist for similar exercises and their development should be looked into in the interest of keeping education in step with technology and its practical application.

APPENDIX A: The Modified TRS-80

The physics department at Rensselaer Polytechnic Institute has been investigating the use of computers in the laboratory for educational purposes for several years. The original concept was to have a central unit running the main software routines and a number of peripheral terminals. This structure permits small groups of students in the freshman laboratory to have simultaneous access to a computer. The initial system implemented was made up of TRS-80 Color Computers. In order to establish the desired network (Figure A.1) some simple hardware modifications were made and software was developed to link the network together.

The hardware modification primarily consisted of "piggy backing" an AM 2732 chip at the U13 location on the circuit board (Figure A.2). This creates a greater memory storage area for down loaded programs. The serial port can be used to transmit the programs to the peripheral stations, however our systems were modified to transmit through the cassette port. This allows the connection of the serial printer to the main unit without disturbing the system. The cartridge port on the central unit is used for the standard peripherals such as disk drives.

In order to send programs to the substations a down loading routine was developed. The resident BASIC language was abandoned in favor of DEFT Pascal Workbench. This language package allows programing in either Pascal or 6809 Assembler language. [16] It can compile either language into a standard object code. Programs and subroutines or procedures can then be linked together and a final program generated in binary machine language. This binary code is then run using the execute command on the TRS-80. This package allows the different parts of a program to be written in the most convenient language and then incorporated into the main program. The entire procedure is executed through the use of a short BASIC program. The following pages list the BASIC language program used to initiate the down loading

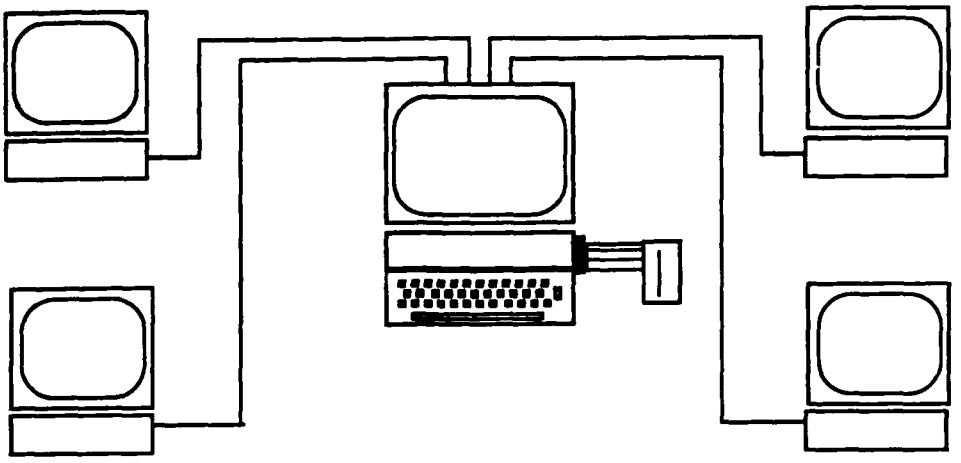


Figure A.1. TRS-80 Network Setup

procedure and the main routine and subroutines written in 6809 Assembler. The resulting program will transmit any binary program from the central unit down to the work stations. Since theoretically it is possible to transmit in either direction some subroutines have been included for the reverse procedure, although they are not used in the down loading program (Dump).

When down loading a program it is possible to leave the resident BASIC in the peripheral computer intact. The following down loading routine overwrites the resident programing (it actually makes it unavailable for use) in order to allow for more efficient screen usage, better graphics and more available program memory. As a result, driving routines for the screen must be sent with the down loaded program or else the monitor will not function as desired. These driver routines include clearing the screen (cls), graphics (plot) and keyboard input and polling. The 6809 Assembler language listings of the routines follow the down load program listings. In order to insure that DEFT Pascal

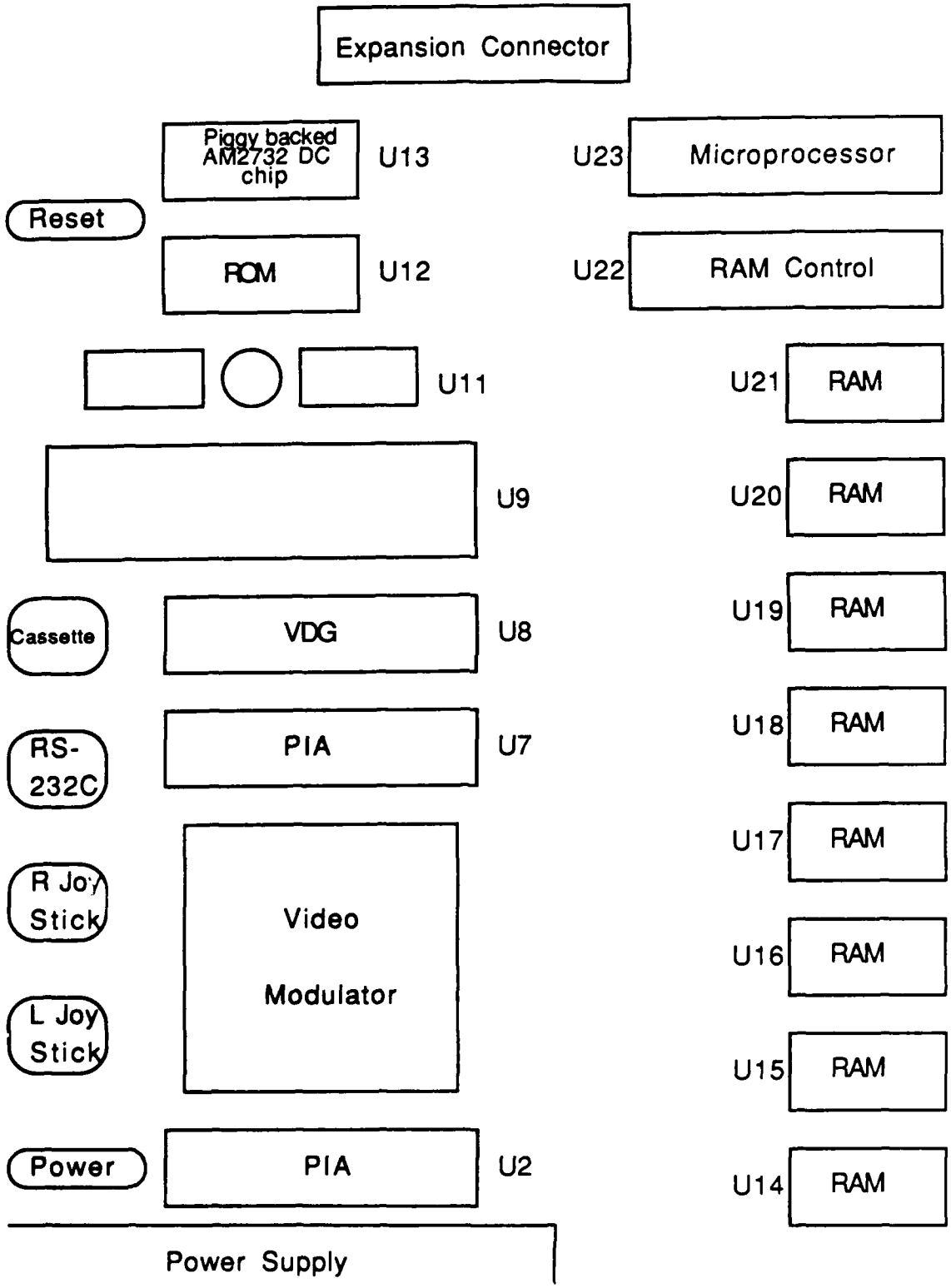


Figure A.2. Color Computer Circuit Layout

recognizes these new commands when compiling code the Pascal library that comes with DEFT Pascal Workbench must be modified. The lines that must be added are at the end of this appendix.

DOWNLOAD INITIATION PROGRAM IN BASIC

```
10 REM $2000-2100 DUMP PROGRAM
20 REM $220-7F00 BUFFER
30 REM
40 PCLEAR 1:CLEAR 50,&H1F00:CLS
50 INPUT"DRIVE";D
60 DRIVE D
70 LINE INPUT"FILENAME:  ";A$
80 A$=A$+".BIN"
90 LOADM A$,&H2000
95 DRIVE 0
100 LOADM "DUMP.BIN"
110 REM DISK OFF
120 POKE &HFF40,0
130 PRINT"HIT ANY KEY TO TRANSFER"
140 IF INKEY$="" THEN 140
150 PRINT"SENDING..."
160 EXEC &H2000:GOTO 130
```

DOWNLOAD PROGRAM IN 6809 ASSEMBLER

```
*
*DOWNLOAD OF PROGRAM
*  $4200-$6200
*
SETUPNET EXT
SETWRITE EXT
WRITEBLK EXT
*
*
 MAIN
START ORCC #$50
 LBSR SETUPNET
 LBSR SETWRITE
 LDX #$4200
 LDY #$6200
 LBSR WRITEBLK
 RTS
 END
```

```
 TITLE /NETWORK SUBROUTINES/
*
* 8/24/86
*
* SUBROUTINES TO HANDLE NETWORK
* FAST VERSION
*
*
 PUBLIC SETUPNET
 PUBLIC SETWRITE
 PUBLIC SETREAD
 PUBLIC SETNONE
 PUBLIC SETBUSY
 PUBLIC CLRBUSY
 PUBLIC READBLK
 PUBLIC WRITEBLK
 PUBLIC NETREAD
 PUBLIC NETWRITE
 PUBLIC NETBUSY
*
*
*
NETSTART RMB 2
NETLENGTH RMB 2
NETERROR RMB 2
*
 PUBLIC NETSTART
 PUBLIC NETLENGTH
 PUBLIC NETERROR
*
*
*
*PORT BITS:
*
* PA7 BUSY IN          0=BUSY
* PA6 DATA IN
* PA5 READ CONTROL     0=ACTIVE
* PA4 WRITE CONTROL    0=ACTIVE
* PA3 DATA OUT
* PA2 BUSY OUT         0=ACTIVE
*
*
*
* NETREAD:
*   NETSTART=START OF BUFFER
*   NETLENGTH=LENGTH OF BUFFER
*
*
*
NETREAD LDX NETSTART,PCR GET START ADDRESS
 LDY NETLENGTH,PCR AND LENGTH
```

```
 BSR READBLK
 STY NETLENGTH,PCR
 CLRA
 STD NETERROR,PCR ERROR CODE
 RTS
*
*
*
*NETWRITE:
*  NETSTART=START ADDRESS
*  NETLENGTH=NUMBER OF BYTES TO WRITE
*
*
NETWRITE LDX NETSTART,PCR GET START ADDRESS
 LDY NETLENGTH,PCR AND LENGTH
 LBSR WRITEBLK
 CLRA
 CLRB
 STD NETERROR,PCR
 RTS
*
*
*
*SETUPNET
*
*   SETS UP PIA REGISTERS
*   DISABLES TRANSMIT,RECEIVE
*
*   EXIT: A,CC CHANGED
*
SETUPNET LDA #$34
 STA $FF20
 LDA $FF21 MAKE DDR
 ANDA #$F8
 STA $FF21
 LDA #$3E II00 000I
 STA $FF20
 LDA #$04 MAKE DR
 ORA $FF21
 STA $FF21
 LDA $FF23 CB2=INPUT
 ANDA #$7 CB2 INTERRUPTS OFF
 STA $FF23
 RTS
*
*
*
SETNONE LDA #$34 INITIAL BITS
 STA $FF20
 RTS
*
*
```

```
* SETWRITE
*
*   SETS 8T28 TO WRITE
*
*   EXIT: A,CC CHANGED
*
SETWRITE LDA $FF20
 ORA #$20
 ANDA #$EF
 STA $FF20
 RTS
*
*
* SETREAD
*
*   SETS 8T28 TO READ
*
*   EXIT: A,CC CHANGED
*
SETREAD LDA $FF20
 ANDA #$DF
 ORA #$10
 STA $FF20
 RTS
*
*
*
* SETBUSY
*
*   PULLS BUSY LINE LOW
*
*   EXIT: A,CC CHANGED
*
SETBUSY LDA $FF20
 ANDA #$FB
 STA $FF20
 RTS
*
*
* CLRBUSY
*
*   RELEASES BUSY LINE
*
*   EXIT: A,CC CHANGED
*
CLRBUSY LDA $FF20
 ORA #$04
 STA $FF20
 RTS
*
*
* FUNCTION NETBUSY : BOOLEAN;
```

```
*
NETBUSY CLR 4,S RETURN VALUE=FALSE
 CLR 5,S
 TST $FF21 CHECK BIT 7
 BMI NB1 1=NOT BUSY
 INC 5,S 0=BUSY
NB1 RTS
*
*
* READBLK
*
*   READS DATA FROM NETWORK; MUST BE IN READ MODE;
*
*   ENTRY: X=BUFFER START ADDRESS
*          Y=BUFFER LENGTH
*
*   EXIT:  A,CC CHANGED
*          X=ADRESS OF LAST BYTE +1
*          Y=# OF BYTES ACTUALLY READ
*
*          Z=1 & B=0: NO ERROR
*
*          Z=0 ERROR:
*            CODE IN B;
*              $FF BUFFER OVERRUN
*
*
READBLK PSHS U,Y,CC SAVE REGISTERS
 BSR SETREAD
 ORCC #$50 NO INTERRUPTS
 LDU #$FF20 U=PORT ADDRESS
*WAIT FOR VALID START OF TRANSMISSION
SYNCS LDY #6
SYNC BSR BYTE GET A BYTE
 CMPA #$16 SYNC?
 BNE SYNCS
 LEAY -1,Y
 BNE SYNC
SYNC1 BSR BYTE WAIT FOR START FLAG
 CMPA #$16
 BEQ SYNC1
 CMPA #2
 BNE SYNCS
*READ LENGTH
 BSR BYTE GET HIGH BYTE
 PSHS D SAVE D
 BSR BYTE GET LOW BYTE
 STA 1,S PUT ON STACK
 PULS Y GET LENGTH
 CMPY 1,S TOO BIG?
 BHI ERR
 STY 1,S SAVE ACTUAL LENGTH
```

```
DATA BSR BYTE GET DATA BYTE
 STA ,X+ STORE IT
 LEAY -1,Y DONE?
 BNE DATA
EXIT LBSR SETNONE DRIVERS OFF
 PULS CC,Y,U
 TSTB SET Z FOR ERROR FLAG
 RTS BYE
ERR LDB #$FF FLAG ERROR
 LDY #0 NO BYTES READ
 BRA EXIT
*
*
* BYTE READS A BYTE
*
*   ENTRY: U=$FF20
*
*   EXIT:  A=NEW BYTE
*          B=0
*          U=$FF20
*          X,Y: UNCHANGED
*
*
BYTE LDD #$0940 A=BIT COUNT B=BIT MASK FOR PORT
 PSHS A
WAIT BITB ,U WAIT FOR START BIT
 BEQ WAIT
 NOP CENTERING DELAY
 NOP
BYTEL LDB ,U GET BIT
 LSLB SHIFT INTO A
 LSLB
 ROLA
 TFR B,B DELAY SOME MORE
 NOP
 NOP
 DEC ,S ANY MORE BITS?
 BNE BYTEL
 PULS B,PC
*
*
*
* WRITEBLK
*
*   SENDS BLOCK OF DATA
*
* FORMAT:
*
*  VALUE  COUNT  EXPLANATION
*
*  $16    10     SYNC BYTES
*
```

```
* $02   1     START INDICATION
*
* $HH   1     HIGH BYTE OF LENGTH
*
* $LL   1     LOW BYTE
*
* $XX   $HHLL   'LENGTH' DATA BYTES
*
*
*   ENTRY: X=START ADDRESS OF BLOCK
*          Y=LENGTH OF BLOCK
*
*   EXIT:  X=END ADDRESS +1
*          D,CC CHANGED
*          Y,U UNCHANGED
*
WRITEBLK PSHS Y,CC SAVE ADDRESS
 LBSR SETWRITE
 ORCC #$50
 LDY #10 SEND 10 SYNC BYTES
WB1 LDA #$16 $16=SYNC BYTE
 BSR OUT
 LDA #70 DELAY ONE BYTE TIME
WBD DECA
 BNE WBD
 LEAY -1,Y
 BNE WB1
 LDA #02 START SEQUENCE
 BSR OUT
 LDA 1,S HIGH ORDER LENGTH
 BSR OUT
 LDA 2,S AND LOW ORDER
 BSR OUT
 LDY 1,S
WBL LDA ,X+ GET BYTE
 BSR OUT
 LEAY -1,Y SEE IF DONE
 BNE WBL
 LBSR SETNONE
 PULS CC,Y,PC
*
*
* OUT SENDS ONE BYTE TO NETWORK
*
*   ENTRY: A=BYTE
*
*   EXIT:  A,B=0
*          CC CHANGED
*
OUT LDB #10 START BIT + 8 DATA + 1 STOP
 PSHS B
 LDB #$08
```

```
 ORB $FF20 PA3(DATA BIT)=1;
 BRA OUT3
OUT1 ASLA NEXT BIT INTO CARRY
 BCC OUT2 IF A ZERO,BRANCH
 ORB #$8 ELSE SET TO 1
 BRA OUT3
OUT2 ANDB #$F7 SET TO ZERO
 BRA OUT3 PRESERVE TIMING
OUT3 STB $FF20 SEND THE BIT
 NOP DELAY SOME
 NOP
 NOP
 DEC ,S DONE?
 BNE OUT1
 PULS B,PC BYE
*
 END
```

```
DRIVER ROUTINES FOR THE MODIFIED TRS-80

*
GSTART EQU 0 START SCREEN
GEND EQU 2 END SCREEN
GY EQU 4
GX EQU 5
GFLAG EQU 6
*
*PUBLIC ROUTINES
*
 PUBLIC CLS PASCAL CALL
 PUBLIC CLRTOP PASCAL CALL
 PUBLIC CURSOR PASCAL CALL
 PUBLIC GSELECT PASCAL
 PUBLIC GSETUP PASCAL CALL
 PUBLIC GSHOW PASCAL/ASM
 PUBLIC PLOT PASCAL CALL
 PUBLIC SETT ASM CALL
 PUBLIC PUTC CALLED BY PATCHED PASIO
 PUBLIC GETCURSOR PASCAL CALL
 PUBLIC INVERSETEXT PASCAL CALL
 PUBLIC VPRINT PASCAL CALL
 PUBLIC GETKEY
 PUBLIC INKEY
*
*EXTERNAL REFERENCES
*
CSET EXT
*
*
* GSELECT
*   SETS GLOBAL POINTER TO GRAPHICS CONTROL BLOCK
*
*   CALL: GSELECT(BLKADD:INTEGER)
*
GSELECT LDX 4,S GET ADDRESS
 STX $88
 RTS
*
* GSETUP(BLKADD,STARTADD,ENDADD:INTEGER)
*
*  BLKADD=ADDRESS OF CONTROL BLOCK
*  STARTADD=ADDRESS OF SCREEN START
*  ENDADD=ADDRESS OF SCREEN END
*         IF ZERO, DEFAULTS TO 6143+STARTADD
*
* STACK FRAME:
BLKADD EQU 8
STARTADD EQU 6
ENDADD EQU 4
```

```
*
GSETUP LDY BLKADD,S
 LDD STARTADD,S
 STD GSTART,Y SAVE START ADDRESS
 LDD ENDADD,S
 BNE GSETUP1
 LDD #6143
 ADDD STARTADD,S
GSETUP1 STD GEND,Y SAVE END ADDRESS
 CLR GX,Y
 CLR GY,Y
 CLR GFLAG,Y
 RTS
*
*
* GSHOW DISPLAYS SCREEN
*
GSHOW LDD [$88] GET START ADDRESS
 BSR SETSAM SET ADDRESS
 STA $FFC3 SET SAM MODE CONTROL REGISTER
 STA $FFC5
 LDA #248 SET VDG CONTROL
 STA 65314
 RTS
*
* SETSAM SETS SAM TO ADDRESS IN A
*            (HIGH 7 BITS)
*      CLEARS GRAPHICS MODE BITS
*
*
SETSAM ANDA #$FE MASK OFF ADDRESS BITS
 LDX #$FFD4
STS1 STA ,--X CLEAR BIT
 CMPX #$FFC0 DONE?
 BEQ STS2
 LSLA GET NEXT BIT
 BCC STS1
 STA 1,X SET SAM BIT IF 1
 BRA STS1
STS2 RTS
*
* CLEAR ROUTINES
*
*
* CLEAR TOP
*
CLRTOP LDD#256
 BRA CLR
*
* CLEAR WHOLE SCREEN
*
CLS LDD #6144
```

```
CLR LDY $88
 LDX GSTART,Y
 LEAX D,X
CLR1 LDD #$FFFF
 TST GFLAG,Y
 BEQ CLR2
 CLRA
 CLRB
CLR2 STD ,--X
 STD ,--X
 STD ,--X
 STD ,--X
 STD ,--X
 STD ,--X
 STD ,--X
 CMPX GSTART,Y
 BHI CLR2
 CLR GY,Y
 CLR GX,Y
 RTS
*
*
* PUTC HANDLES CHARACTER OUTPUT
*
*
PUTC PSHS D,X,Y,U SAVE THINGS
 LDY $88 GET POINTER
 ANDA #$7F MAKE <128
 CMPA #$20 CONTROL CHAR.?
 BLO CONTROLCHAR
 LBSR PUTCOK DISPLAY IT
 LDB GX,Y UPDATE CURSOR;
 ADDB #5 ADD ONE CHAR. WIDTH
 BCS PUTC1 CRLF IF PAST END
 CMPB #251 OR TOO FAR RIGHT
 BLS PUTC2
PUTC1 BSR CRLF
 PULS D,X,Y,U,PC BYE
PUTC2 STB GX,Y UPDATE X
 PULS D,X,Y,U,PC BYE
*
CONTROLCHAR CMPA #$0D CR?
 BNE NOCRLF
 BSR CRLF DO CRLF
 PULS D,X,Y,U,PC
*
NOCRLF CMPA #$0A LF:
 BNE NOLF
 BSR LF DO LF
 PULS D,Y,X,U,PC
*
```

```
NOLF CMPA #8 BACKSPACE?
 BNE PUTCEXIT NO, INVALID CHAR
* DO BACKSPACE
 LDD GY,Y BACK UP ONE CHAR.
 CMPB #5 BEGINNING OF LINE?
 BHS BACK1
 CMPA #9 TOP OF SCREEN?
 BLO PUTCEXIT
 SUBA #9 NO, BACK UP ONE LINE
 CLRB  TO COLUMN 251
BACK1 SUBB #5 BACK UP ONE WIDTH
 STD GY,Y
 LDA #$20 PRINT SPACE
 BSR PUTCOK
PUTCEXIT PULS D,X,Y,U,PC BYE
*
CRLF CLR GX,Y X=0 DOES CR
LF LDA GY,Y Y+9 DOES LF
 ADDA #9
 CMPA #191-9 PAST END?
 BHI SCROLL IF YES, SCROLL IT
 STA GY,Y
 RTS
*
* SCROLL
*
SCROLL PSHS D,X,Y,U SAVE THINGS
 LDX GSTART,YSTART
 LEAU 32*9,X ONE LINE DOWN
 LDY GEND,Y END ADDRESS
 PSHS Y
 TFR X,Y
*
* PARTIALLY UNFOLLED LOOP
* MOVES ONE SCAN LINE (32 BYTES)
* AT A TIME
*
SCROLLOOP PULU D,X GET 4 BYTES
 STD ,Y++ STORE THEM
 STX ,Y++
 PULU D,X ETC
 STD ,Y++
 STX ,Y++
 PULU D,X
 STD ,Y++
 STX ,Y++
 PULU D,X
 STD ,Y++
 STX ,Y++
 PULU D,X
 STD ,Y++
 STX ,Y++
```

```
 PULU D,X
 STD ,Y++
 STX ,Y++
 PULU D,X
 STD ,Y++
 STX ,Y++
 PULU D,X
 STD ,Y++
 STX ,Y++
 CMPU ,S SEE IF DONE
 BLO SCROLLOOP
 LDD #$FFFF CLEAR BOTTOM LINE
SCRL1 STD ,Y++
 STD ,Y++
 STD ,Y++
 STD ,Y++
 CMPY ,S SEE IF DONE
 BLO SCRL1
 LEAS 2,S CLEAR STACK
 PULS D,Y,X,U,PC BYE
*
* PUTCOK DRAWS CHARACTER
*
*  ENTRY:
*        Y=ADDRESS OF DESCRIPTOR
*        A=CHARACTER
*
*  STACK USAGE:
*        3,S BYTE COUNT
*       1-2,S STORAGE
*        0,S SHIFT VALUE
*
PUTCOK SUBA #$20 OFFSET FOR NON-PRINTING CHARACTERS
 LDB #$8 BYTES PER CHAR
 PSHS B LOOP COUNTER
 MUL INDEX TO CHARACTER
 LEAU CSET,PCR BASE OF CHARACTERS
 LEAU D,U ADDRESS OF CHAR
 LBSR SETT GET PIXEL ADDRESS,BIT
 LSLB ADJUST BIT SO MUL DOES SHIFT
 PSHS B,X SAVE SHIFT FACTOR,RESERVE 2 BYTES
 CMPX BEND,Y PAST END?
 BHS PUTCO3
PUTCO2 LDA ,U+ GET CHAR BYTE
 LDB ,S GET SHIFT FACTOR
 BEQ PUTCNS DON'T SHIFT IF ALREADY OK
 MUL SHIFT CHAR BYTE
PUTCNS STD 1,S STORE SHIFTED VALUE
 LDA #$F8 GET MASK BYTE
 LDB ,S GET SHIFT FACTOR
 BEQ PUTCNS1
 MUL SHIFT MASK
```

```
PUTCNS1 TST GFLAG,Y
 BEQ NORMCHAR
 COMA FLIP MASK
 COMB
 ANDA ,X RESET BITS OF BACKGROUND
 ANDB 1,X
 BRA BITCHAR
NORMCHAR ORA ,X SET BITS OF BACKGROUND
 ORB 1,X
BITCHAR EORA 1,S FLIP BITS CHAR
 EORB 2,S
 STD ,X STORE IT ON SCREEN
 LEAX $20,X NEXT ROW
 CMPX BEND,Y OFF SCREEN?
 BHS PUTCO3
 DEC 3,S ANY MORE BYTES?
 BNE PUTCO2
PUTCO3 PULS D,X,PC
*
*
* PLOT
*
*   SETS/RESETS PIXEL
*
*  CALL: PLOT(X,Y,FLAG :INTEGER)
*
*  PASSED:
*        B,S: X LOW
*        A,S: X HIGH
*        9,6: Y LOW
*        8,S: Y HIGH
*        7,S: MODE 0=DOT ON
*        6,S: HIGH MODE (UNUSED)
*       4-5,S: LINK ADDRESS
*       2-3,S: RETURN ADDRESS
*       0-1,S: SAVED GX,GY
*
*  CC CHANGED
*
PLOT LDY $88 GET POINTER
 LDD GY,Y
 PSHS D
 LDA 9,S A=Y COORD
 LDB $B,S B=X COORD
 STD BY,Y
 BSR SETT
 CMPX GEND,Y
 BHI DOTEND
 TST 7,S
 BEQ DOTCLEAR
DOTSET ORB ,X
 BRA DOTPUT
```

```
DOTCLEAR COMB
 ANDB ,X
DOTPUT STB ,X
DOTEND PULS D
 STD GY,Y
 RTS
*
*
* CURSOR
*
*   MOVES CURSOR TO X,Y
*
* CALL: CURSOR(X,Y:INTEGER)
*
* STACK FRAME:
*           7,S: X LOW
*           6,S: X HIGH
*           5,S: Y LOW
*           4,S: Y HIGH
*         2-3,S: LINK ADDRESS
*         0-1,S: RETURN ADDRESS
*
CURSOR LDY $88
 LDB 7,S
 LDA 5,S
 STD GY,Y
 RTS
*
* SETT SETS UP
*
*  ENTRY:
*       GY=Y
*       GX=X
*       Y=POINTER TO CONTROL BLOCK
*  EXIT:
*       BYTE ADDRESS IN X
*       BIT # IN A
*       BIT IN B
*
SETT LDD GY,Y
 LSRA
 RORB
 LSRA
 RORB
 LSRA
 RORB
 ADDD GSTART,Y
 TFR D,X
 LDA GX,Y
 ANDA #7
 PSHS Y
 LEAY SHFTB,PCR
```

```
 LDB A,Y
 PULS Y,PC
SHFTB FDB $8040
 FDB $2010
 FDB $0804
 FDB $0201
*
*
* GETCURSOR RETURNS POSITION TO PASCAL
*
* CALL: GETCURSOR(VAR X,Y:INTEGER)
*
* STACK FRAME:
*          6-7,S: ADDRESS OF X
*          4-5,S: ADDRESS OF Y
*          2-3,S: LINK ADDRESS
*          0-1,S: RETURN ADDRESS
*
GETCURSOR LDY $88
 CLRA
 LDB GY,Y
 STD [4,S]
 LDB GX,Y
 STD [6,S]
 RTS
*
*
* INVERSETEXT: SETS INVERSE FLAG TO VALUE FROM PASCAL
*
* CALL: INVERSETEXT(FLAG:BOOLEAN)
*
* STACK FRAME:
*           5,S: BOOLEAN FLAG
*           4,S: UNUSED
*         2-3,S: LINK
*         0-1,S: RETURN ADDRESS
*
INVERSETEXT LDA 5,S
 LDY $88
 STA GFLAG,Y
 RTS
*
*
* VPRINT
*
*  PRINTS VERTICAL STRING
*
*  CALL: VPRINT(S:STRING)
*
*  STACK FRAME:
*           5-N,S: STRING
*             4,S: LENGTH
```

```
*           2-3,S: LINK
*           0-1,S: RETURN ADDRESS
*
VPRINT LDY $68
 LDB GX,Y
 CMPB #251 CHECK X COORD
 BLS VP1
 LDB #251
VP1 STB GX,Y
 LDB 4,S GET LENGTH
 LEAX 5,S GET POINTER TO STRING
 PSHS U SAVE DYNAMIC LINK (FOR PASCAL)
 PSHS B,X SAVE STRING INFO
VPL LDA GY,Y MOVE PRINT LOCATION
 CMPA #183 AT BOTTOM OF SCREEN?
 BLS VPL1
 LDA #183
VPL1 ADDA #9 DOWN 9 COLUMNS
 STA GY,Y STORE IT
 PULS B,X GET INFO
 TSTB SEE IF DONE
 BEQ VPEND
 DECB
 LDA ,X+ GET CHAR
 PSHS B,X SAVE INFO
 LBSR PUTCOK
 BRA VPL
VPEND PULS U,PC
*
DFTPOLLKEY EXT
*
*
* PASCAL CALL
*
* FUNCTION INKEY:CHAR
*
* WAITS FOR A KEY TO BE PRESSED
* RETURNS ASCII CODE OF KEY
*
* STACK:
*      5,S: RETURN FUNCTION BYTE
*    2-3,S: LINK
*    0-1,S: RETURN ADDRESS
*
INKEY LBSR DFTPOLLKEY
 BEQ INKEY
 STA 5,S
 RTS
*
* PASCAL CALL
*
* FUNCTION GETKEY:CHAR;
```

```
*
* SCANS KEYBOARD ONCE
* RETURNS ZERO IF NO KEY
*
GETKEY LBSR DFTPOLLKEY
 STA 5,S
 RTS
*
 END
```

CHARACTER SET FOR DRIVER SUBROUTINES

```
 PUBLIC CSET
CSET FCB $00,$00,$00,$00,$00,$00,$00,$00
 FCB $20,$20,$20,$20,$20,$00,$20,$00
 FCB $50,$50,$50,$00,$00,$00,$00,$00
 FCB $50,$50,$F8,$50,$F8,$50,$50,$00
 FCB $20,$38,$60,$30,$28,$70,$20,$00
 FCB $68,$68,$10,$20,$20,$58,$58,$00
* FCB $20,$50,$50,$20,$68,$50,$28,$00
 FCB $20,$70,$A8,$20,$20,$A8,$70,$20
 FCB $30,$20,$40,$00,$00,$00,$00,$00
 FCB $10,$20,$40,$40,$40,$20,$10,$00
 FCB $20,$10,$08,$08,$08,$10,$20,$00
 FCB $00,$48,$30,$78,$30,$48,$00,$00
 FCB $00,$20,$20,$F8,$20,$20,$00,$00
 FCB $00,$00,$00,$00,$30,$30,$20,$40
 FCB $00,$00,$00,$78,$00,$00,$00,$00
 FCB $00,$00,$00,$00,$00,$30,$30,$00
 FCB $08,$08,$10,$20,$20,$40,$40,$00
 FCB $30,$48,$48,$48,$48,$48,$30,$00
 FCB $20,$60,$20,$20,$20,$20,$70,$00
 FCB $30,$48,$08,$10,$20,$40,$78,$00
 FCB $30,$48,$08,$30,$08,$48,$30,$00
 FCB $08,$18,$28,$48,$78,$08,$08,$00
 FCB $78,$40,$70,$08,$08,$48,$30,$00
 FCB $10,$20,$40,$70,$48,$48,$30,$00
 FCB $78,$08,$08,$10,$20,$40,$40,$00
 FCB $30,$48,$48,$30,$48,$48,$30,$00
 FCB $30,$48,$48,$38,$08,$10,$20,$00
 FCB $00,$30,$00,$00,$30,$00,$00,$00
 FCB $00,$30,$00,$00,$30,$10,$20,$00
 FCB $08,$10,$20,$40,$20,$10,$08,$00
 FCB $00,$00,$78,$00,$78,$00,$00,$00
 FCB $40,$20,$10,$08,$10,$20,$40,$00
 FCB $30,$48,$08,$10,$20,$00,$20,$00
* FCB $30,$48,$58,$58,$40,$48,$30,$00
 FCB $F8,$40,$20,$10,$20,$40,$F8,$00
 FCB $30,$48,$48,$78,$48,$48,$48,$00
 FCB $70,$48,$48,$70,$48,$48,$70,$00
 FCB $30,$48,$40,$40,$40,$48,$30,$00
 FCB $70,$48,$48,$48,$48,$48,$70,$00
 FCB $78,$40,$40,$70,$40,$40,$78,$00
 FCB $78,$40,$40,$70,$40,$40,$40,$00
 FCB $30,$48,$40,$58,$48,$48,$38,$00
 FCB $48,$48,$48,$78,$48,$48,$48,$00
 FCB $70,$20,$20,$20,$20,$20,$70,$00
 FCB $08,$08,$08,$08,$08,$48,$30,$00
 FCB $48,$48,$50,$60,$50,$48,$48,$00
 FCB $40,$40,$40,$40,$40,$40,$78,$00
 FCB $48,$78,$78,$48,$48,$48,$48,$00
```

```
FCB $48,$68,$68,$58,$58,$48,$48,$00
FCB $78,$48,$48,$48,$48,$48,$78,$00
FCB $70,$48,$48,$70,$40,$40,$40,$00
FCB $30,$48,$48,$48,$68,$58,$38,$00
FCB $70,$48,$48,$70,$50,$48,$48,$00
FCB $30,$48,$40,$30,$08,$48,$30,$00
FCB $70,$20,$20,$20,$20,$20,$20,$00
FCB $48,$48,$48,$48,$48,$48,$30,$00
FCB $48,$48,$48,$48,$48,$30,$30,$00
FCB $48,$48,$48,$48,$78,$78,$48,$00
FCB $48,$48,$30,$30,$30,$48,$48,$00
FCB $88,$88,$50,$20,$20,$20,$20,$00
FCB $78,$08,$10,$20,$20,$40,$78,$00
FCB $78,$60,$60,$60,$60,$60,$78,$00
FCB $40,$40,$20,$20,$10,$08,$08,$00
FCB $78,$18,$18,$18,$18,$18,$78,$00
FCB $20,$70,$A8,$20,$20,$20,$20,$00
FCB $00,$10,$20,$78,$20,$10,$00,$00
FCB $60,$60,$20,$10,$00,$00,$00,$00
FCB $00,$00,$38,$08,$38,$48,$38,$00
FCB $40,$40,$70,$48,$48,$48,$70,$00
FCB $00,$00,$38,$40,$40,$40,$38,$00
FCB $08,$08,$38,$48,$48,$48,$38,$00
FCB $00,$00,$30,$48,$78,$40,$38,$00
FCB $10,$28,$20,$70,$20,$20,$20,$00
FCB $00,$00,$30,$48,$48,$38,$08,$30
FCB $40,$40,$70,$48,$48,$48,$48,$00
FCB $20,$00,$60,$20,$20,$20,$70,$00
FCB $10,$00,$10,$10,$10,$10,$50,$20
FCB $40,$40,$40,$50,$60,$50,$48,$00
FCB $60,$20,$20,$20,$20,$20,$70,$00
FCB $00,$00,$78,$78,$78,$48,$48,$00
FCB $00,$00,$40,$70,$48,$48,$48,$00
FCB $00,$00,$30,$48,$48,$48,$30,$00
FCB $00,$00,$70,$48,$48,$70,$40,$40
FCB $00,$00,$38,$48,$48,$38,$08,$08
FCB $00,$00,$58,$60,$40,$40,$40,$00
FCB $00,$00,$38,$40,$30,$08,$70,$00
FCB $20,$20,$70,$20,$20,$28,$10,$00
FCB $00,$00,$48,$48,$48,$48,$38,$00
FCB $00,$00,$48,$48,$48,$30,$30,$00
FCB $00,$00,$48,$48,$48,$78,$78,$00
FCB $00,$00,$48,$48,$30,$48,$48,$00
FCB $00,$00,$48,$48,$48,$38,$08,$30
FCB $00,$00,$78,$08,$10,$20,$78,$00
FCB $10,$20,$20,$40,$20,$20,$10,$00
FCB $20,$20,$20,$20,$20,$20,$20,$00
FCB $40,$20,$20,$10,$20,$20,$40,$00
FCB $48,$B0,$00,$00,$00,$00,$00,$00
FCB $50,$A8,$50,$A8,$50,$A8,$50,$A8
END
```

REQUIRED ADDITIONS TO PASCALIB/EXT FROM DEFT

```
TYPE GBLK1 = RECORD
      STARTADD :INTEGER;
      ENDADD: INTEGER;
      GY : CHAR;
      GX: CHAR;
      INVERSEFLAG: BOOLEAN;
    END;

PROCEDURE GSELECT (VAR BLK: GBLK1);

PROCEDURE GSETUP (VAR BLK: GBLK1; STARTADD,ENDADD: INTEGER);

PROCEDURE GSHOW;

PROCEDURE CLS;

PROCEDURE CLRTOP;

PROCEDURE CURSOR (X,Y: INTEGER);

PROCEDURE GETCURSOR (VAR X,Y: INTEGER);

PROCEDURE PLOT (X,Y,MODE: INTEGER);

PROCEDURE INVERSETEXT (FLAG: BOOLEAN);

PROCEDURE VPRINT (S: STRING);

PROCEDURE GETKEY: CHAR;

PROCEDURE INKEY: CHAR;
```

APPENDIX B: Operating Programs

The routines written for these experiments are primarily in Pascal. The programs for the TRS-80 Color Computer were written with DEFT Pascal Workbench and any differences or embellishments to ISO Standard Pascal are in the cited manuals. The Physpend program was written with Turbo Pascal from Borland International and its variations from ISO Standard Pascal are also documented. [17] Commenting on the TRS-80 routines has been minimized due to limited memory, however logic flow charts precede the listings as necessary. Standard conventions on "pretty printing" have been used for readability. The common subroutines were written in 6809 Assembler and commented accordingly.

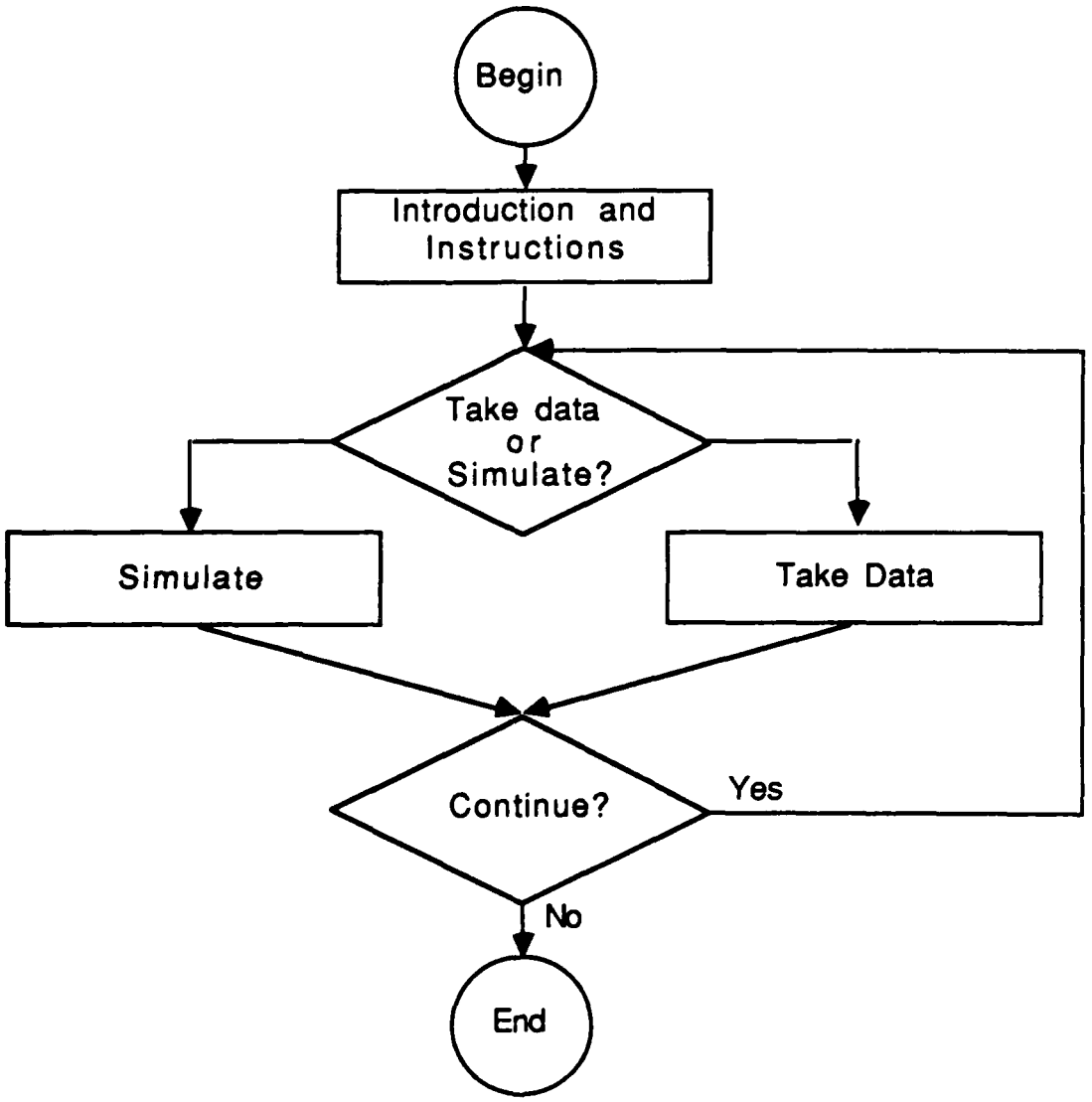


Figure B.1. Freshpen and Airwheel Main Routine

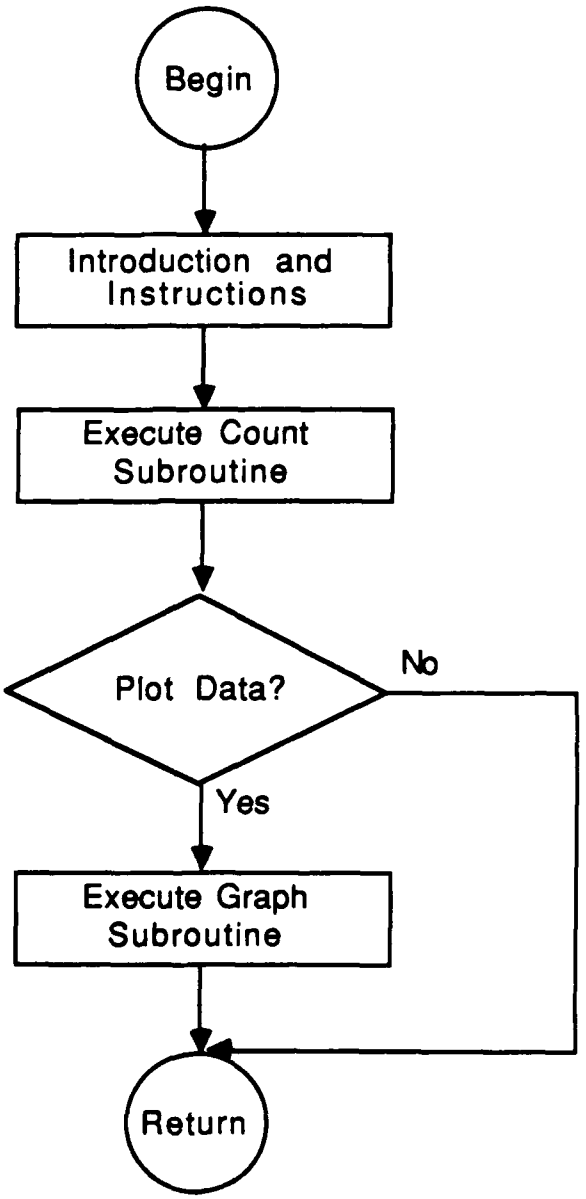


Figure B.2. Takedata Procedure

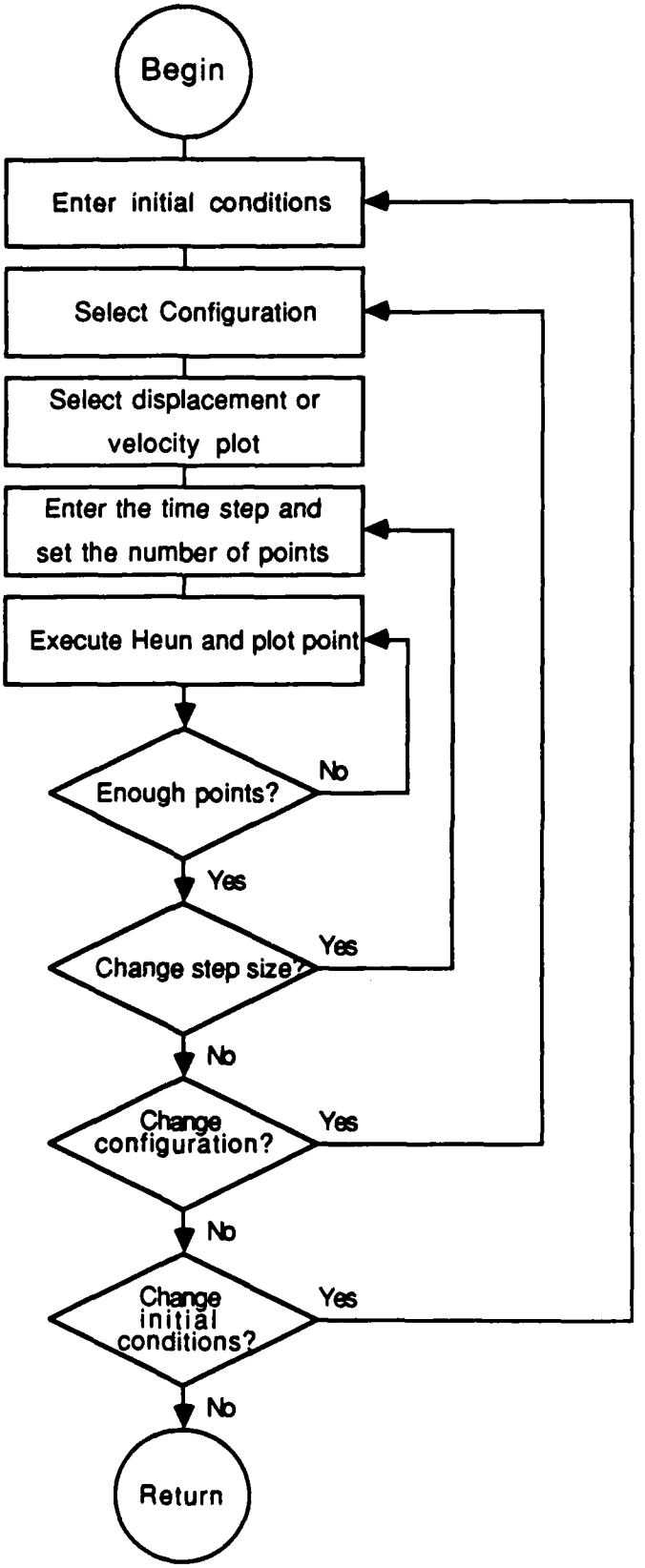


Figure B.3. Freshpen Simulate Procedure

```
PROGRAM Freshpendulum (input, output);

 LABEL 1;

 TYPE
  vector = ARRAY [1..2] OF real;
  datapoints = ARRAY [1..11] OF vector;
  point = ARRAY [1..2] OF integer;
  data = ARRAY [1..64] OF integer;
  simdata = ARRAY [1..256] OF point;

 VAR
  select : char;
  buffer : data;

{************************************************************}

PROCEDURE Setup ; EXTERNAL;

PROCEDURE Count (address : integer);  EXTERNAL;

PROCEDURE Graph (address : integer);  EXTERNAL;

{************************************************************}

FUNCTION Yes : Boolean;

 VAR
  ch : char;

 BEGIN
  readln (ch);
  IF ch = 'Y' THEN Yes := true ELSE Yes := false
 END;

{************************************************************}

PROCEDURE Noprime (pendconst : real; VAR upass, uprime :
                   vector);

 BEGIN
  uprime[1] := upass[2];
  uprime[2] := - pendconst * sin(upass[1]);
 END;

{************************************************************}

PROCEDURE Lprime (pendconst, airlength, airconst : real;
                  VAR upass, uprime : vector);
```

```
 BEGIN
  uprime[1] := upass[2];
  uprime[2] := - pendconst * sin(upass[1]) - airconst *
                 airlength * upass[2];
 END;

{*********************************************************}

PROCEDURE Sqprime (pendconst, airlength, airconst : real;
                   VAR upass, uprime : vector);

 BEGIN
  uprime[1] := upass[2];
  IF upass[2] < 0
   THEN
    uprime[2] := - pendconst * sin(upass[1]) +
                 airconst * sqr(airlength * upass[2])
   ELSE
    uprime[2] := - pendconst * sin(upass[1])
                 - airconst * sqr(airlength * upass[2])
 END;

{*********************************************************}

PROCEDURE Takedata (VAR buffer : data);

 VAR
  i : integer;
  inchar : char;

 BEGIN
  cls;
  writeln ('Hit any key to start timing.');
  inchar := Inkey;
  Clrtop;
  writeln ('Collecting . . .');
  Count (Buffer[1]);
  writeln ('Do you wish to see the data points? (Y/N)');
  IF Yes THEN
   BEGIN
    writeln ('Hit any key when you are finished with');
    writeln ('the data.');
    writeln;
    writeln ('Are you ready to plot? (Y/N)');
    IF Yes THEN
     BEGIN
      cls;
      Graph(Buffer[1]);
      inchar := Inkey;
     END;
   END; { then }
 END;
```

```
{*********************************************************}

PROCEDURE Heun (pendconst, airlength, airconst deltat:real;
                VAR thetadot : datapoints; VAR i : integer;
                acase : char);

  VAR
    k1, k2, utemp : vector;
    j : integer;

  BEGIN
    CASE acase OF
      'N' :
        BEGIN
          utemp[1] := thetadot[i,1];
          utemp[2] := thetadot[i,2];
          Noprime (pendconst, utemp, k1);
          FOR j := 1 TO 2 DO
            utemp[j] := thetadot[i,j] + deltat * k1[j];
          Noprime (pendconst, utemp, k2);
          FOR j := 1 TO 2 DO
            thetadot[i+1,j] := thetadot[i,j] + deltat *
                               (k1[j] + k2[j])/2;
        END; { case of N }
      'L' :
        BEGIN
          utemp[1] := thetadot[i,1];
          utemp[2] := thetadot[i,2];
          Lprime (pendconst,airlength,airconst,utemp,k1);
          FOR j := 1 TO 2 DO
            utemp[j] := thetadot[i,j] + deltat * k1[j];
          Lprime (pendconst,airlength,airconst,utemp,k2);
          FOR j := 1 TO 2 DO
            thetadot[i+1,j] := thetadot[i,j] + deltat *
                               (k1[j] + k2[j])/2;
        END; { case of L }
      'Q' :
        BEGIN
          utemp[1] := thetadot[i,1];
          utemp[2] := thetadot[i,2];
          Sqprime (pendconst,airlength,airconst,utemp,k1);
          FOR j := 1 TO 2 DO
            utemp[j] := thetadot[i,j] + deltat * k1[j];
          Sqprime (pendconst,airlength,airconst,utemp,k2);
          FOR j := 1 TO 2 DO
            thetadot[i+1,j] := thetadot[i,j] + deltat *
                               (k1[j] + k2[j])/2;
        END { case of Q }
    END; { case }
  END; { Heun }
```

```
{************************************************************}

PROCEDURE Simulate (buffer : data);

  LABEL 1,2,3,4,5,6,7;

  CONST
    g = 9.81;
    conv = 0.34906585;

  VAR
    acase, curve, inchar : char;
    mass, cmdistance, imoment theta1, airlength airconst,
    pendconst, deltat : real;
    i, j, subint, int : integer;
    thetadot : datapoints;
    theta, omega : simdata;

  BEGIN
    cls;
    int := 0;
  7: writeln ('Enter the mass of the pendulum.');
    readln (mass);
    writeln ('Enter the distance to the center of mass.');
    readln (cmdistance);
    writeln ('Enter the moment of inertia.');
    readln (imoment);
    pendconst := mass * g * cmdistance / imoment;
    cls;
    writeln ('In conducting the simulation you may');
    writeln ('neglect air resistance, include it with');
    writeln ('a term linear in velocity or a term');
    writeln ('quadratic in velocity.  Enter: ');
    writeln ('      N - Neglect air resistance');
    writeln ('      L - Include it linearly');
    writeln ('      Q - Include quadratic term');
    readln (acase);
    IF NOT (acase IN ['N','L','Q']) THEN
      BEGIN
        writeln ('Not a proper response.');
        GOTO 1;
      END;
  2: cls;
    writeln ('Enter the initial angle (rad).');
    readln (theta1);
    IF acase IN ['L','Q'] THEN
      BEGIN
        writeln ('Enter the distance from the pivot to ');
        writeln ('the center of the plates.');
        readln (airlength);
        writeln ('Enter the coefficient of air');
        writeln ('resistance.');
```

```
      readln (airconst);
    END;
5: cls;
  writeln ('Do you wish to plot the displacement or ');
  writeln ('velocity?  Enter:');
  writeln ('      D - Displacement');
  writeln ('      V - Velocity');
  readln (curve);
  IF NOT (curve IN ['D','V']) THEN
    BEGIN
      writeln ('Not a proper response.');
      GOTO 5;
    END;
6: writeln('Enter the step size, delta t, to be used in');
  writeln ('the simulation.  (Multiples of .025 or ');
  writeln ('divisors of .025 down to .0025 only for ');
  writeln ('graphing purposes.)');
  readln (deltat);
  writeln('The simulation will plot the data, the last');
  writeln ('simulation and one point at a time for the');
  writeln ('new simulation.  Hit S if you wish to stop');
  writeln ('the plot.  After you are fininshed viewing');
  writeln ('the plot hit any key to continue the ');
  writeln ('program.');
3: writeln
  writeln('Are you ready to plot the simulation? (Y/N)');
  IF Yes THEN
    BEGIN
      cls;
      Graph (Buffer[1]);
      FOR k := 1 TO int DO
        BEGIN
          IF curve = 'V' THEN
            Plot (omega[i,1],omega[i,2],0)
          ELSE
            Plot (theta[i,1],theta[i,2],0);
        END;
      IF deltat < .025 THEN
        BEGIN
          subint := round(.025/deltat);
          int := 255;
        END
      ELSE
        BEGIN
          subint := 1;
          int := round(6.375/deltat);
        END;
      theta[1,1] := 0;
      omega[1,1] := 0;
      theta[1,2] := 190 - round(theta1*100);
      omega[1,2] := 190;
      CASE curve OF
```

```
        'D' :  Plot(theta[1,1],theta[1,2],0);
        'V' :  Plot(omega[1,1],omega[1,2],0)
      END; { case }
      thetadot[1,1] := theta1;
      thetadot[1,2] := 0.0;
      FOR i := 1 TO int DO
        BEGIN
          FOR j:= 1 TO subint DO
            Heun (pendconst,airlength,airconst,deltat,
                 thetadot,j,acase);
          theta[i+1,2] := 190 - round(abs(thetadot
                                [subint+1,1]) * 100);
          omega[i+1,2] := 190 - round(abs(thetadot
                                [subint+1,2])/conv *3);
          IF subint = 1 THEN
            BEGIN
              theta[i+1,1] := round(i*deltat/.025);
              omega[i+1,1] := theta[i+1,1];
            END
          ELSE
            BEGIN
              theta[i+1,1] := i;
              omega[i+1,1] := i;
            END;
          thetadot[1,1] := thetadot[subint+1,1];
          thetadot[1,2] := thetadot[subint+1,2];
          CASE curve OF
            'D':  Plot(theta[i+1,1],theta[i+1,2],0);
            'V':  Plot(omega[i+1,1],omega[i+1,2],0)
          END; { case }
          Plot(omega[i+1,1],5,0); { mark progress }
          IF Getkey = 'S' THEN GOTO 4;
        END;  { for }
    END { then }
  ELSE
    GOTO 3;
4: inchar :- Inkey;
  cls;
  writeln ('Do you wish to change the step size? (Y/N)');
  IF Yes
    THEN GOTO 6;
  writeln ('Do you wish to change the initial air');
  writeln ('resistance constants? (Y/N)');
  IF Yes
    THEN GOTO 2;
  writeln ('Do you wish to change the way that air');
  writeln ('resistance is considered? (Y/N)');
  IF Yes
    THEN GOTO 1;
  writeln ('Do you wish to change the mass or moment');
  writeln ('of inertia? (Y/N)');
  IF Yes
```

```
    THEN GOTO 7;
  END; { Simulate }

{*********************************************************}

  BEGIN { Freshpendulum }
    Setup;
    cls;
    writeln ('Welcome to the pendulum datataking and ');
    writeln ('comparison program.  This program will ');
    writeln ('allow you to determine the best way to');
    writeln('incorporate air resistance in the equations');
    writeln ('of motion and an approximate value for the');
    writeln ('coefficient of air resistance.  You may ');
    writeln ('also investigate the accuracy of the ');
    writeln ('numerical method as you change the size ');
    writeln ('of the step in time.');
    writeln;
  1: writeln ('Do you wish to take data or run a ');
    writeln ('simulation?  Enter:');
    writeln ('     T - Takedata');
    writeln ('     S - Simulate');
    readln (select);
    CASE select OF
      'T':  Takedata (buffer);
      'S':  Simulate (buffer)
      ELSE
        BEGIN
          writeln ('Not a proper response.');
          GOTO 1;
        END
    END: { case }
    cls;
    writeln ('Do you wish to continue the program? (Y/N)');
    IF Yes THEN
      GOTO 1;
  END. { Freshpendulum }
```

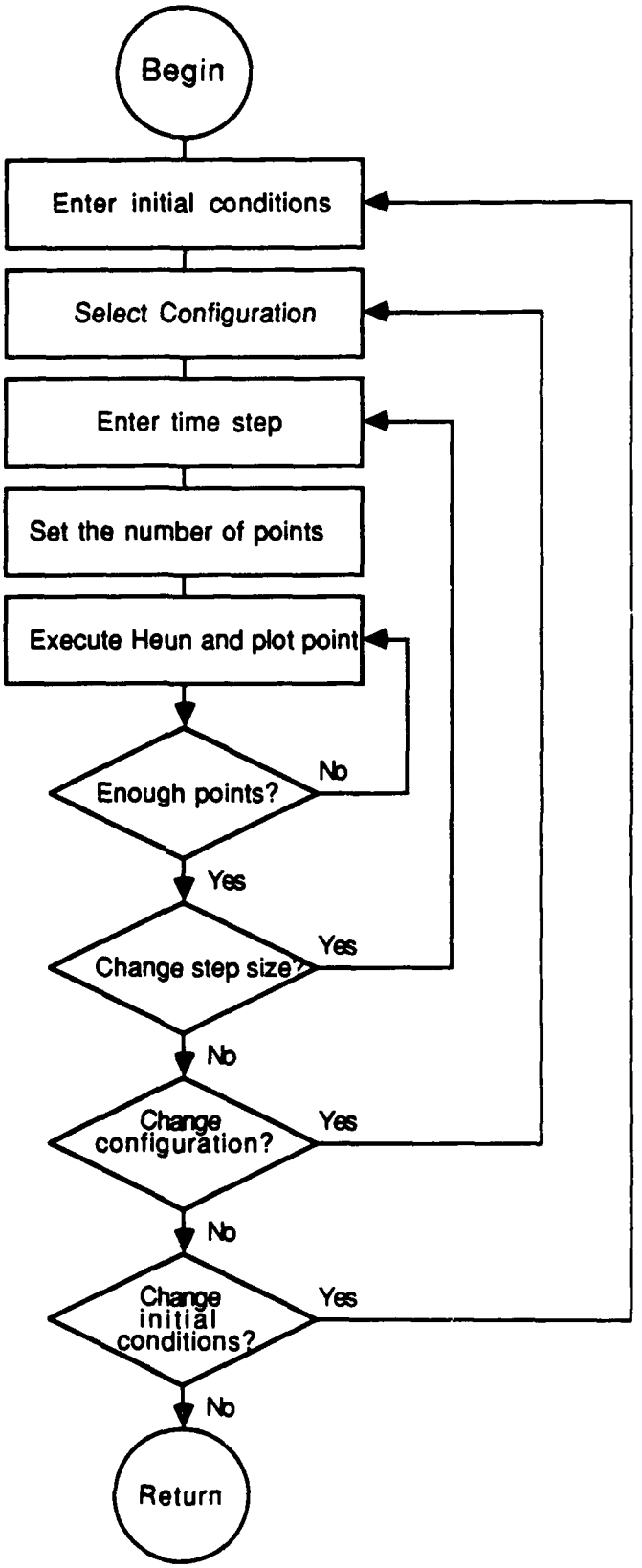


Figure B.4. Airwheel Simulate Procedure

```
PROGRAM Airwheel (input,output);

 LABEL 1;

 TYPE
  vector = ARRAY[1..2] OF real;
  calcvals = ARRAY[1..11] OF vector;
  point = ARRAY[1..2] OF integer;
  data = ARRAY[1..64] OF integer;
  simdata = ARRAY[1..256] OF point;

 VAR
  select : char;
  omega : simdata;
  buffer : data;

{************************************************************}

PROCEDURE Setup;  EXTERNAL;

PROCEDURE Count (address : integer);  EXTERNAL;

PROCEDURE Graph (address : integer);  EXTERNAL;

{************************************************************}

FUNCTION Yes : boolean;

 VAR
  ch : char;

 BEGIN
  readln (ch);
  IF ch = 'Y' THEN Yes := true ELSE Yes := false
 END;

{************************************************************}

PROCEDURE Noprime (wheelconst : real; VAR upass, uprime :
                   vector);

 BEGIN
  uprime[1] := upass[2];
  uprime[2] := wheelconst;
 END;

{************************************************************}

PROCEDURE Lprime (wheelconst, airlength, airconst, imoment:
                  real; VAR upass, uprime : vector);
```

```
  BEGIN
    uprime[1] := upass[2];
    uprime[2] := wheelconst - airconst * airlength/imoment
                 * upass[2];
  END;

{*********************************************************}

PROCEDURE Sqprime(wheelconst, airlength, airconst, imoment:
                  real; VAR upass, uprime : vector);

  BEGIN
    uprime[1] := upass[2];
    uprime[2] := wheelconst - airconst/imoment *
                   sqr(airlength * upass[2]);
  END;

{*********************************************************}

PROCEDURE Takedata (VAR buffer : data);

  VAR
    i : integer;
    inchar : char;

  BEGIN
    cls;
    writeln ('Hit any key to start timing');
    inchar := Inkey;
    clrtop;
    writeln (Collecting . . .');
    Count (Buffer[1]);
    writeln ('Do you wish to see the data points? (Y/N)');
    IF Yes THEN
      BEGIN
        writeln ('Hit any key when you are finished with');
        writeln ('the data.');
        writeln;
        writeln ('Are you ready to plot? (Y/N)');
        IF Yes THEN
          BEGIN
            cls;
            Graph (Buffer[1]);
            inchar := Inkey;
          END;
      END; { then }
  END; { takedata }

{*********************************************************}

PROCEDURE Heun (wheelconst, airlength, airconst, imoment,
                deltat : real; VAR thetadot : calcvals;
```

```
        VAR i : integer, acase : char);

VAR
  k1, k2, utemp : vector;
  j : integer;

BEGIN
  CASE acase OF
    'N' :
      BEGIN
        utemp[1] := thetadot[i,1];
        utemp[2] := thetadot[i,2];
        Noprime (wheelconst, utemp, k1);
        FOR j := 1 TO 2 DO
          utemp[j] := thetadot[i,j] + deltat * k1[j];
        Noprime (wheelconst, utemp, k2);
        FOR j := 1 TO 2 DO
          thetadot[i+1,j] := thetadot[i,j] + deltat *
                             (k1[j]+ k2[j])/2;
      END; { case of N }
    'L' :
      BEGIN
        utemp[1] := thetadot[i,1];
        utemp[2] := thetadot[i,2];
        Lprime (wheelconst, airlength, airconst, imoment,
              utemp, k1);
        FOR j := 1 TO 2 DO
          utemp[j] := thetadot[i,j] + deltat * k1[j];
        Lprime (wheelconst, airlength, airconst, imoment,
              utemp, k2);
        FOR j := 1 TO 2 DO
          thetadot[i+1,j] := thetadot[i,j] + deltat *
                             (k1[j]+ k2[j])/2;
      END; { case of L }
    'Q' :
      BEGIN
        utemp[1] := thetadot[i,1];
        utemp[2] := thetadot[i,2];
        Sqprime (wheelconst, airlength, airconst, imoment,
              utemp, k1);
        FOR j := 1 TO 2 DO
          utemp[j] := thetadot[i,j] + deltat * k1[j];
        Sqrime (wheelconst, airlength, airconst, imoment.
              utemp, k2);
        FOR j := 1 TO 2 DO
          thetadot[i+1,j] := thetadot[i,j] + deltat *
                             (k1[j]+ k2[j])/2;
      END; { case of Q }
  END; { case }
END; { Heun }

{*************************************************************}
```

```
PROCEDURE Simulate (buffer : data);

 LABEL 1,2,3,4,5,6;

 CONST
   g = 9.81;
   conv = 0.34906585;

 VAR
   inchar, acase : char;
   mass, aradius, imoment, airlength, airconst,wheelconst,
   deltat : real;
   i, j, subin, int : integer;
   thetadot : calcvals;
   omega : simdata;

 BEGIN
   cls;
   int := 0;
   writeln ('The simulation solves the equation of ');
   writeln ('motion for the accelerated wheel using');
   writeln ('Heuns method.');
   writeln;
6: writeln ('Enter the moment of inertia for the wheel.');
   readln (imoment);
   writeln ('Enter the radius of the disk to the cord.');
   readln (aradius);
   writeln ('Enter the mass of the accelerating weight.');
   readln (mass);
   wheelconst := mass*g*aradius/(imoment+mass*sqr(aradius));
   cls;
   writeln ('In conducting the simulation you may ');
   writeln ('neglect air resistance, include it with a');
   writeln('term linear in velocity or a term quadratic');
   writeln ('in velocity.  Enter:');
1: writeln ('     N - Neglect air resistance');
   writeln ('     L - Include it Linearly');
   writeln ('     Q - Include the Quadratic term');
   readly (acase);
   IF NOT (acase IN ['N','L','Q']) THEN
     BEGIN
       writeln ('Not a proper response.');
       GOTO 1;
     END;
2: cls;
   IF acase IN ['L','Q'] THEN
     BEGIN
       writeln ('Enter the radius to the center of the');
       writeln ('plates.');
       readln (airlength);
       writeln ('Enter the coefficient of air');
```

```
      writeln ('resistance.');
      readln (airconst);
    END;
5: writeln('Enter the step size, delta t, to be used in');
   writeln ('the simulation.  (Multiples of .025 or ');
   writeln ('divisors of .025 down to .0025 only for ');
   writeln ('graphing purposes.)');
   readln (deltat);
   writeln ('The simulaton will plot the data, the last');
   writeln('simulation and one point at at time for the');
   writeln ('new simulation.  Hit S if you wisht to ');
   writeln ('stop the plot.  After you are finished ');
   writeln ('viewing the plot hit any key to continue');
   writeln ('the program.');
3: writeln;
   writeln('Are you ready to plot the simulation? (Y/N)');
   IF Yes THEN
    BEGIN
      cls;
      Graph (Buffer[1]);
      FOR i := 1 TO int DO
        Plot(omega[i,1],omega[i,2],0);
      IF deltat < .025 THEN
        BEGIN
          subint := Round(.025/deltat);
          int := 255;
        END
      ELSE
        BEGIN
          subint := 1;
          int := Round(6.375/deltat);
        END;
      omega[1,1] := 0;
      omega[1,2] := 190;
      Plot(0,190,0);
      thetadot[1,1] := 0.0;
      thetadot[1,2] := 0.0;
      FOR i := 1 TO int DO
        BEGIN
          FOR j := 1 TO subint DO
            Heun (wheelconst,airlength,airconst,imoment,
                  deltat, thetadot, j, acase);
          omega[i+1,2] := 190 - Round(abs
                        (thetadot[subint+1,2])/conv * 4);
          IF subint = 1
            THEN omega[i+1,1] := Round(i*deltat/.025)
            ELSE omega[i+1,1] := i;
          thetadot[1,1] := thetadot[subint+1,1];
          thetadot[1,2] := thetadot[subint+1,2];
          Plot(omega[i+1,1],omega[i+1,2],0);
          Plot(omega[i+1,1],5,0); { mark progress }
          IF Getkey = 'S' THEN GOTO 4;
```

```
        END; { for }
      END { then }
    ELSE
      GOTO 3;
4: inchar := Inkey;
   cls;
   writeln ('Do you wish to change the initial step ');
   writeln ('size? (Y/N)');
   IF Yes
     THEN GOTO 5;
   writeln ('Do you wish to change the initial air');
   writeln ('resistance constants? (Y/N)');
   IF Yes
     THEN GOTO 2;
   writeln ('Do you wish to change the way air ');
   writeln ('resistance is considered? (Y/N)');
   IF Yes
     THEN GOTO 1;
   writeln ('Do you wish to change the values for the');
   writeln ('moment of inertia or the accelerating ');
   writeln ('weight? (Y/N)');
   IF Yes
     THEN GOTO 6;
 END; { simulate }

{************************************************************}

 BEGIN { Airwheel }
   Setup;
   cls;
   writeln ('Welcome to the rotational motion lab.');
   writeln ('This program will allow you to measure');
   writeln ('the rotational velocity of a wheel with a');
   writeln ('given moment of inertia when under a');
   writeln ('constant torque (provided by gravity). The');
   writeln ('effects of air resistance can also be ');
   writeln ('measured. After data is taken a simualtion');
   writeln ('may be run to investigate the moment of');
   writeln ('inertia of the system, and the dependence');
   writeln ('of the torque due to air resistance on the');
   writeln ('velocity.  You will also be able to ');
   writeln ('investigate the accuracy of the numerical');
   writeln ('method as you change the size of the step');
   writeln ('in time.');
   writeln;
 1: writeln ('Do you wish to take data or run the');
   writeln ('simulation?  Enter:');
   writeln ('     T - Takedata');
   writeln ('     S - Simulate');
   readln (select);
   CASE select OF
     'T' : Takedata (buffer);
```

```
    'S' : Simulate (buffer)
    ELSE
      BEGIN
        writeln ('Not a proper response.');
        GOTO 1;
      END
  END; { case }
  cls;
  writeln ('Do you wish to continue the program? (Y/N)');
  IF Yes
    THEN GOTO 1;
END. { Airwheel }
```

```
PROGRAM Jpendata (input, output);

  LABEL 1;

  TYPE
    vector = ARRAY[1..2 OF real;
    datapoints = ARRAY[1..65] OF vector;
    data = ARRAY[1..64] OF integer;

  VAR
    buffer : data;
    omega : datapoints;
    inchar : char;

{*********************************************************}

PROCEDURE Setup;  EXTERNAL;

PROCEDURE Count (address : integer);  EXTERNAL;

{*********************************************************}

PROCEDURE Takedata (VAR buffer : data);

  BEGIN
    cls;
    writeln ('Hit any key to start timing.');
    inchar := Inkey;
    clrtop;
    writeln ('Collecting . . .');
    Count(Buffer[1]);
  END; { Takedata }

{*********************************************************}

PROCEDURE Convertdata (buffer: data;VAR omega: datapoints);

  CONST
    conv = 0.34906585;

  VAR
    i : integer;

  BEGIN
    cls;
    writeln ('Converting data . . .');
    omega[1,1] := 0.0;
    omega[1,2] := 0.0;
    FOR i := 1 TO 64 DO
      BEGIN
        omega[i+1,1] := 0.1 * i;
```

```
      omega[i+2,2] := buffer[i]/4 * conv
    END; { for }
  END; { Convertdata }

{*****************************   *************************}

PROCEDURE Outputdata (omega : datapoints);

  LABEL 1,2;

  VAR
    outfile : text;
    inchar, select : char;
    start, stop, i : integer;

  BEGIN
    cls;
  1: writeln ('What output device do you want to use? ');
    writeln ('Enter:');
    writeln ('     P - Printer');
    writeln ('     S - Screen');
    readln (select);
    IF NOT (select IN ['P','S']) THEN
      BEGIN
        writeln ('Not a proper response.');
        GOTO 1;
      END;
    CASE select OF
      'P':
        BEGIN
          rewrite (outfile,':-2');
          writeln (outfile,'    Time            Omega');
          FOR i := 1 TO 65 DO
            writeln (outfile,omega[i,1]:12,'     ',
                     omega[i,2]:12);
          close (outfile);
        END; { case of P }
      'S':
        BEGIN
          writeln ('The data will be written one screen');
          writeln ('at a time.  When you are finished ');
          writeln('with a screen hit any key to view the');
          writeln ('next screen of data points.');
          writeln;
          writeln ('Hit any key to start the output.');
          inchar := Inkey;
          start := 1;
          stop := 16;
        2: cls;
          writeln ('    Time            Omega');
          FOR i := start TO stop DO
            writeln (omega[i,1]:12,'     ',omega[i,2]:12);
```

```
        inchar := Inkey;
        start := stop + 1;
        stop := stop + 16;
        IF stop <= 64 THEN GOTO 2;
      END { case of S }
    END; { case }
  END; { Outputdata }

{*************************************************************}

 BEGIN  { Jpendata }
   cls;
   Setup;
   writeln ('This routine collects the data from the');
   writeln ('pendulum, converts and outputs it for');
   writeln ('manual input into the Zenith program.');
   writeln;
   writeln ('Hit any key to proceed.');
   inchar := Inkey;
 1: Takedata (buffer);
   Convertdata (buffer, omega);
   Outputdata (omega);
   writeln ('Do you wish to take another set of data?');
   writeln ('(Y/N)');
   inchar := Inkey;
   IF inchar = 'Y' THEN GOTO 1;
 END. { Jpendata }
```

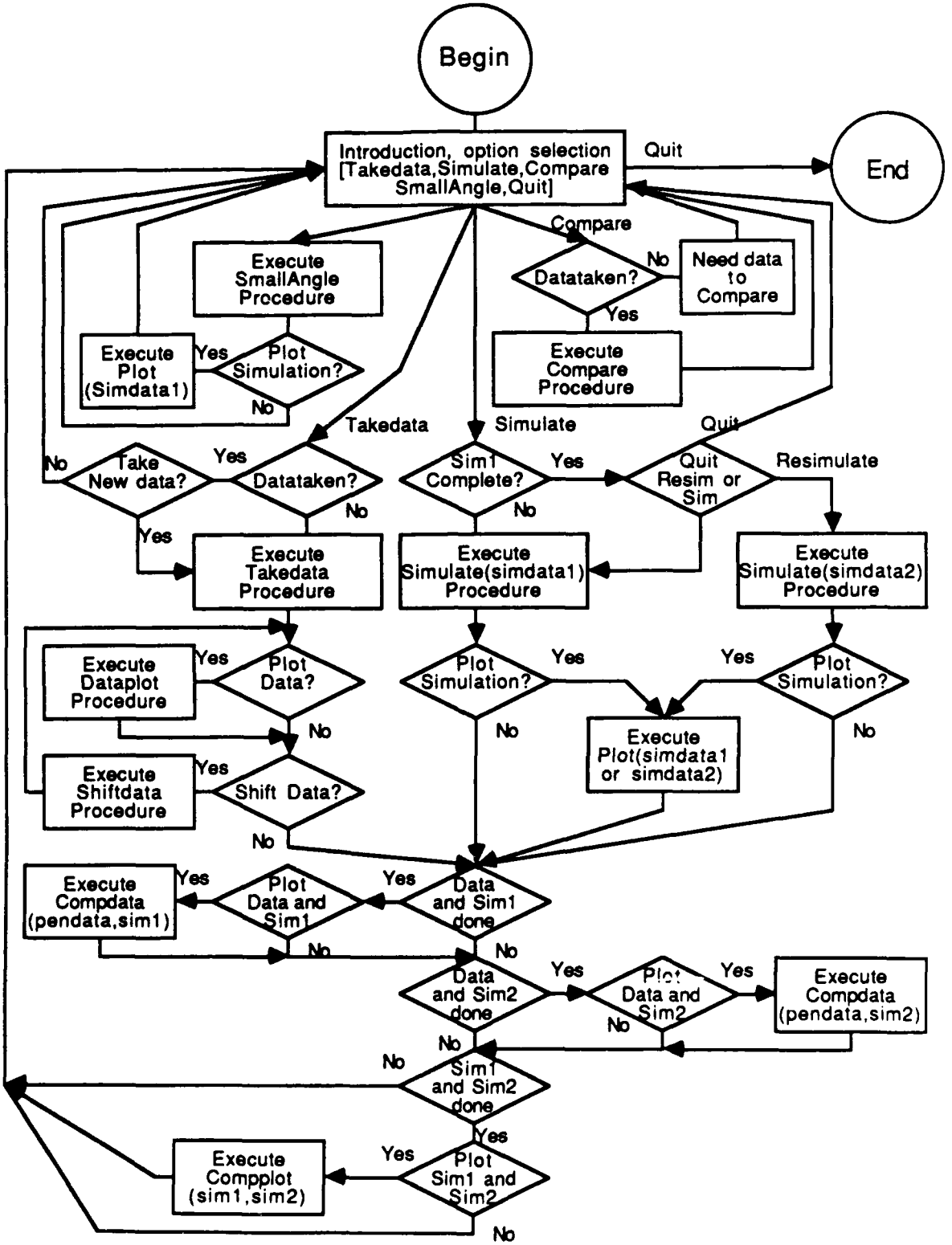


Figure B.5. Physpend Main Routine

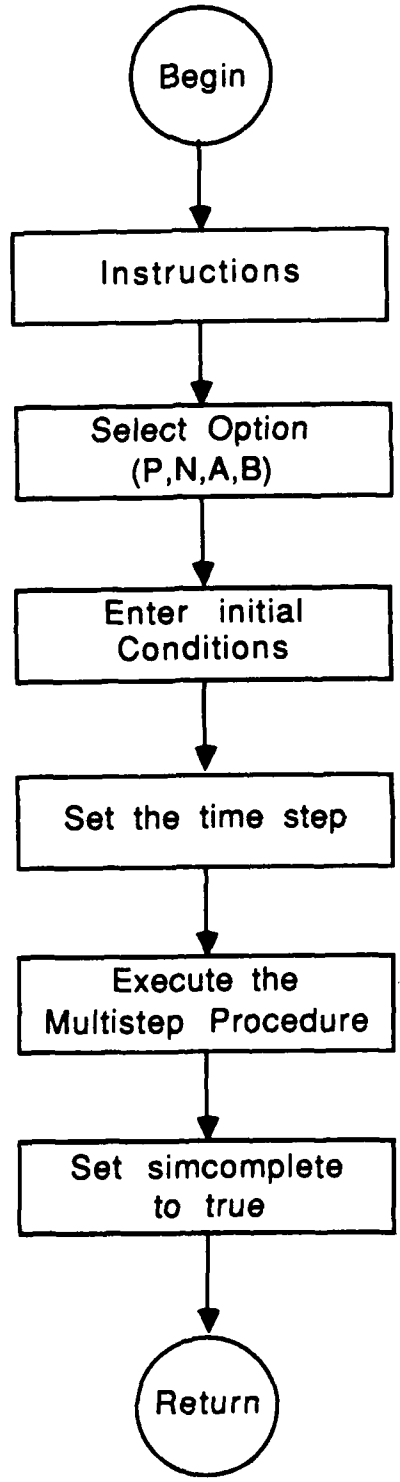


Figure B.6. Physpend Simulate Procedure

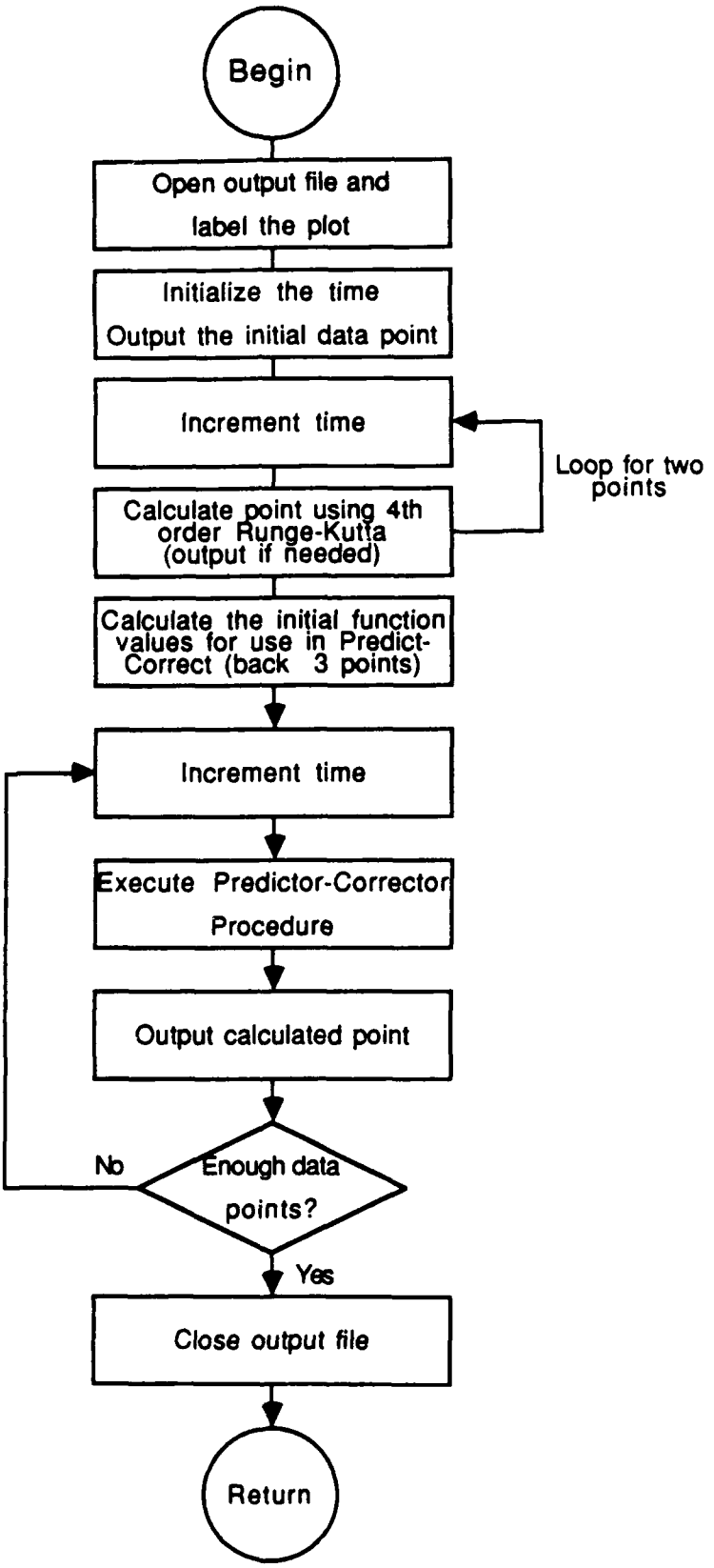


Figure B.7. Procedure Multistep

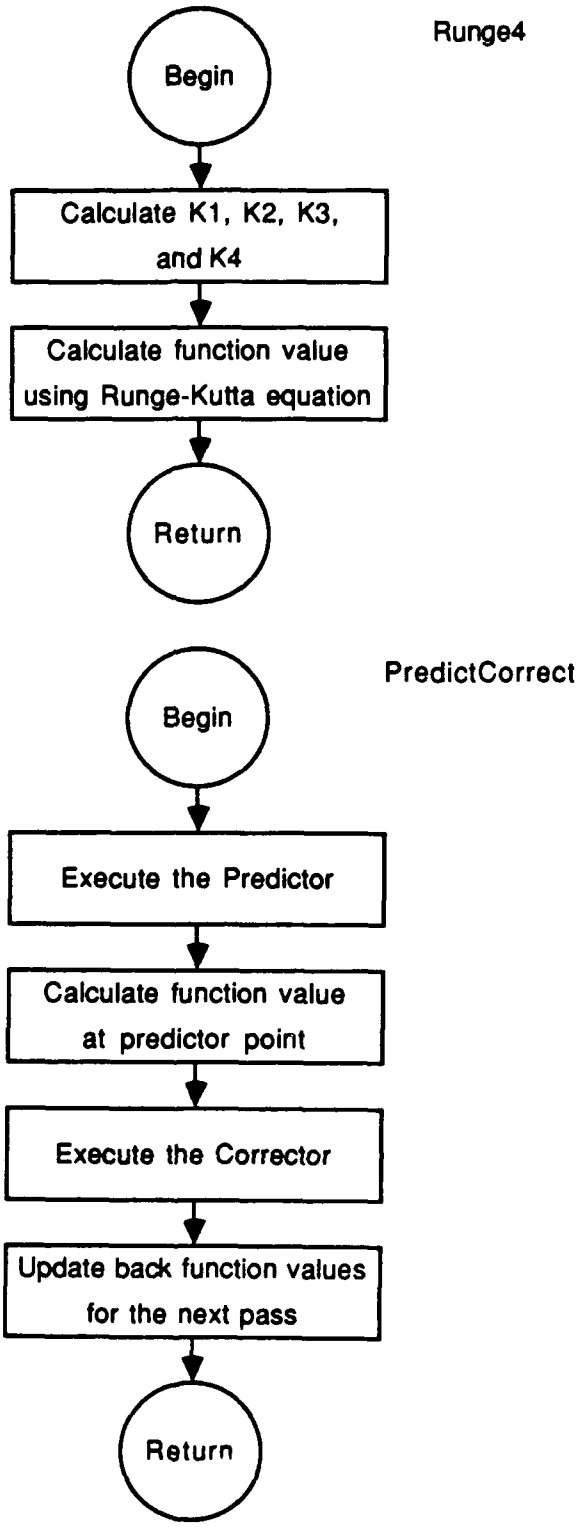


Figure B.8. Procedure Runge4 and PredictCorrect

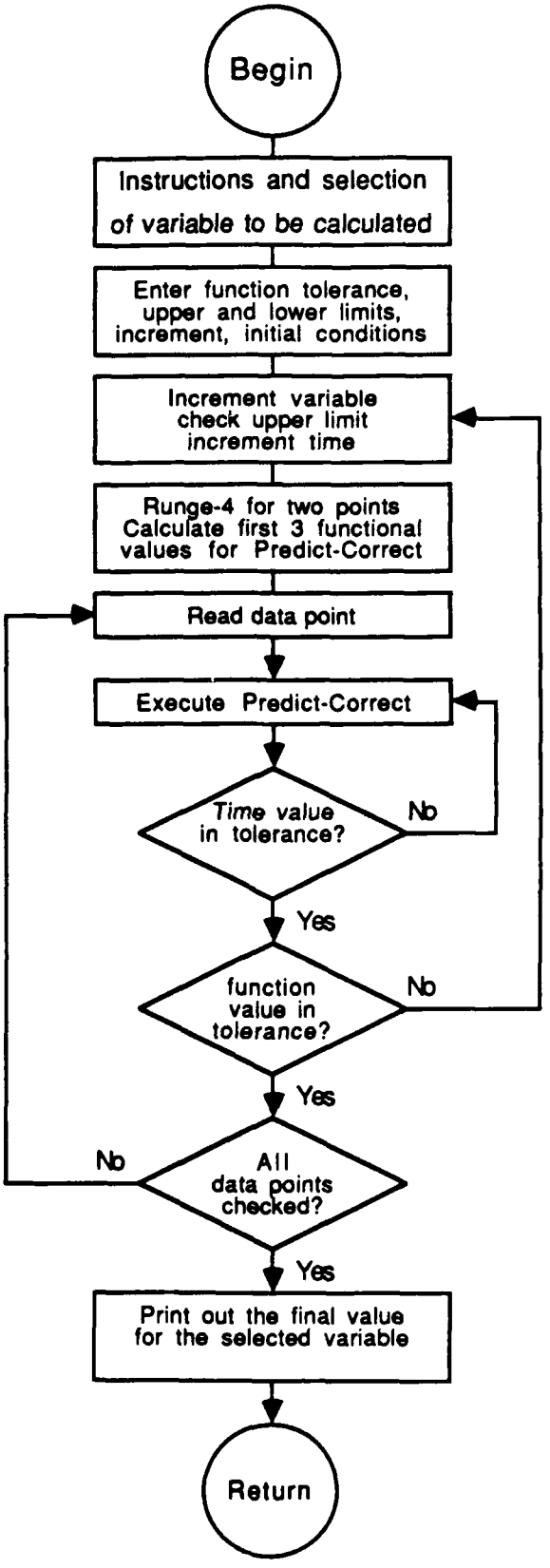


Figure B.9. Physpend Compare Procedure

```
{ This is the physical pendulum data collection and
  simulation routine for the laboratory. The Takedata
  procedure requires input by hand as it is very dependent
  on the physical set up and hardware used in the lab.
  This program is designed to simulate the motion of a
  physical pendulum treating it as a simple pendulum or
  physical pendulum under no constraints, air resistance,
  friction or a combination. Comparison of the motion
  using the small angle approximation is also possible.
  An additional feature allows the calculation of the
  moment of inertia, friction coefficient or air
  resistance coefficient for an arbitrary physical
  pendulum by fitting a simulated curve for the system to
  the actual data using an iterative process. The code
  that follows was developed on an AT® PC6300 using Turbo
  Pascal. The laboratory system uses a TRS-80 Color
  Computer and DEFT Pascal Workbench to take data and a
  Zenith PC and Turbo Pacal to run this software.
  This code was written by Mark R. Stevens. }


PROGRAM PhysPendulum (input, output, simdata, pendata);

  LABEL 1, 2, 3;

  TYPE
    vector = ARRAY [1..2] OF real;
    pastpoints = ARRAY [1..4] OF vector;
    datafile = FILE OF vector;

  VAR
    response, simselect : char;
    datataken, sim1complete, sim2complete : boolean;
    simdata1, simdata2, pendata : datafile;

{************************************************************}

FUNCTION Yes : boolean;

{ This function is used whenever a response is requested of
  a yes or no nature. It is used extensively throughout
  the program and procedures. }

  VAR
    ch : char;

  BEGIN
    readln (ch);
    IF ch IN ['y','Y'] THEN Yes := true ELSE Yes := false
  END;
```

```
{*********************************************************}

PROCEDURE Takedata (VAR datataken : boolean);

{ This procedure may be modified for direct input.
  Currently it is set up to accept inputed datafrom the
  keyboard.  64 data points, specifically. }

  LABEL 1;

  VAR
    omega : vector;
    select : char;
    i : integer;

  BEGIN
    writeln ('Do you wish to use the data currently on ');
    writeln ('the disk? (Y/N)');
    IF Yes THEN GOTO 1;
    rewrite (pendata);
    omega[1] := ord('V');{ Label data as a velocity plot. }
    omega[2] := 0.0;
    write (pendata, omega);
    writeln ('Currently the pendulum data must be input ');
    writeln ('by hand.  First input the configuration of');
    writeln('the pendulum.  Are you expecting to neglect');
    writeln('friction, consider air resistance, etc.');
    writeln('Enter:');
    writeln ('     N - No constraints');
    writeln ('     A - Air resistance only');
    writeln ('     F - Friction effects only');
    writeln ('     B - Both friction and air resistance');
    readln (select);
    omega[1] := ord(select);
    writeln ('Enter the pendulum constant (m*g*d/I), if');
    writeln ('known, otherwise enter zero.');
    readln (omega[2]);
    write (pendata, omega);
    writeln ('If air resistance is considered, enter the');
    writeln ('length to the center of the plates.');
    readln (omega[1]);
    writeln ('Enter the initial angle.');
    readln (omega[2]);
    write (pendata,omega);
    omega[1] := 0.0; { These two data spots are reserved }
    omega[2] := 0.0; { for airconst and frictconst which }
    write (pendata,omega); { are unknowns for the data }
    writeln ('Enter the data points one point at a time,');
    writeln ('with time first followed by the value for');
    writeln ('omega separated by a space.  Continue ');
    writeln ('entering data until all 64 points are ');
    writeln ('entered.  You should also change the sign');
```

```
    writeln ('of omega when it should be negative as the');
    writeln ('Radio Shack computer ouputs the absolute ');
    writeln ('values.  If the initial angle is positive');
    writeln ('the first values of omega should be');
    writeln ('negative until they cross the axis.');
    FOR i := 1 TO 64 DO
      BEGIN
        readln (omega[1], omega[2]);
        write (pendata, omega);
      END;
1 : datataken := true;
    close (pendata)
  END;

{************************************************************}

PROCEDURE Shiftdata (VAR pendata : datafile);

  VAR
    temp : vector;
    tshift : real;

  BEGIN
    reset (pendata);
    clrscr;
    writeln('Shifting the data is possible in both');
    writeln('directions.  A shift of more than a few');
    writeln('tenths of a second is not recommended.  It');
    writeln('would be better to take new data.  Note:');
    writeln('one pixel on the screen is .01 seconds.');
    writeln;
    writeln('Enter the shift for the data(include sign).');
    readln (tshift);
    seek (pendata, 5);
    WHILE NOT eof(pendata) DO
      BEGIN
        read (pendata, temp);
        temp[1] := temp[1] + tshift;
        seek (pendata, FilePos(pendata) - 1);
        write (pendata, temp);
      END;
    close (pendata);
  END; { shiftdata }

{************************************************************}

PROCEDURE Axis;

  BEGIN
    draw (10, 10, 10, 189, 5);
    draw (10, 110, 620, 110, 5);
  END;
```

```
{*********************************************************}

PROCEDURE DataPlot (VAR plotdata : datafile);

{ This procedure takes whatever datafile it is given and
  plots the data with crosses and with labels which are
  included in the first three vector values.}

  LABEL 1;

  VAR
    x1, y1, x2, y2, config : integer;
    buff, temp, temp2 : vector;
    curve : char;

  BEGIN
    clrscr;
    graphmode;
    hires;
    Axis;
    reset (plotdata);
    read (plotdata, temp);
    config := round(temp[1]);
    curve := chr(config);
    read (plotdata, temp, buff,temp2);
    config := round(temp[1]); { Get labeling data }
    writeln('Data Config = ',chr(config),' m*g*d/I = ',
            temp[2]:5:3,'Airlength = ',buff[1]:5:3,
            ' Theta(0) = ',buff[2]:5:3);
    WHILE NOT eof(plotdata) DO
      BEGIN
        read (plotdata, buff);
        x2 := round(buff[1] * 100) + 10;
        y2 := 110 - round(buff[2] * 10);
        draw (x2, y2-2, x2, y2+2, 5);
        draw (x2-4, y2, x2+4, y2, 5);
      END; { while }
    close (plotdata);
1 : writeln ('Are you done with the plot? (Y/N)');
    IF Yes
      THEN textmode
      ELSE GOTO 1
  END;

{*********************************************************}

PROCEDURE Plot (VAR plotdata : datafile);

{ This procedure takes the datafile it is given and plots
  it with labels which are included in the first four
  vector values. }
```

```
LABEL 1;

VAR
  x1, y1, x2, y2, config, scale : integer;
  buff, temp, temp2 : vector;
  curve : char;

BEGIN
  clrscr;
  graphmode;
  hires;
  Axis;
  reset (plotdata);
  read (plotdata, temp);
  config := round(temp[1]);
  curve := chr(config);
  IF curve  IN ['v','V'] THEN
    BEGIN
      scale := 10;
      writeln ('Velocity plot');
    END
  ELSE
    BEGIN
      scale := 40;
      writeln ('Displacement');
    END;
  read (plotdata, temp, buff,temp2);
  config := round(temp[1]); { Get labeling data }
  writeln('Config = ',chr(config),' m*g*d/I = ',
          temp[2]:5:3,' Airlength = ',buff[1]:5:3,
          ' Theta(0) = ',buff[2]:5:3);
  writeln('          Frict Coef = ',temp2[1]:5:3,
          ' Air Coef = ',temp2[2]:5:3);
  read (plotdata, buff);
  x2 := round(buff[1] * 100) + 10;
  y2 := 110 - round(buff[2] * scale);
  WHILE NOT eof(plotdata) DO
    BEGIN
      x1 := x2;
      y1 := y2;
      read (plotdata, buff);
      x2 := round(buff[1] * 100) + 10;
      y2 := 110 - round(buff[2] * scale);
      draw (x1, y1, x2, y2, 5);
    END; { while }
  close (plotdata);
1 : writeln ('Are you done with the plot? (Y/N)');
  IF Yes
    THEN textmode
    ELSE GOTO 1
END;
```

```
{*********************************************************}

PROCEDURE Smallangle (VAR simdata : datafile);

{ This procedure utilizes the small angle approximation to
  calculate the motion of a physical pendulum under no
  constraints.  It allows comparison simulations to be
  calculated to the more exact numerical solution.  Such
  comparisons identify when the small angle approximation
  is a reasonable assumption and when it is not. }

  CONST
    g = 9.81;
    delta_t = 0.01;

  VAR
    time, imoment, mass, itheta, theta, cmdistance,
    pconst : real;
    buffer, temp : vector;
    i : integer;
    curve : char;

  BEGIN
    writeln;
    writeln('Enter the pendulum mass in kilograms.');
    readln (mass);
    writeln ('Enter the distance from the pivot to the');
    writeln ('center of mass.');
    readln (cmdistance);
    writeln('Enter the moment of inertia for the system.');
    readln (imoment);
    writeln ('Enter the initial angle in radians.');
    readln (itheta);
    pconst := mass * g * cmdistance / imoment;
    rewrite (simdata);
    writeln('Do you wish to plot displacement or ');
    writeln ('velocity?  Enter:');
    writeln ('     D - Displacement');
    writeln ('     V - Velocity');
    readln (curve);
    buffer[1] := ord(curve);
    buffer[2] := 0.0;
    write (simdata, buffer);
    buffer[1] := ord('N'); { Label data }
    buffer[2] := pconst;
    temp[1] := 0.0;
    temp[2] := itheta;
    write (simdata, buffer, temp);
    temp[2] := 0.0;
    write (simdata,temp);{ zero values for frictconst and }
    time := 0.0;          { airconst }
```

```
    buffer[1] := time;
    buffer[2] := itheta;
    write (simdata, buffer);
    FOR i := 1 TO 640 DO
      BEGIN
        time := time + delta_t;
        buffer[1] := time;
        IF curve IN ['d','D'] THEN
          buffer[2] := itheta * cos(sqrt(pconst) * time)
        ELSE
          buffer[2] := -itheta * sin(sqrt(pconst) * time)
                       * sqrt(pconst);
        write (simdata, buffer);
      END; { for i }
    close (simdata);
  END; { Smallangle }

{************************************************************}

PROCEDURE Noprime(pendconst: real;VAR upass,uprime:vector);

{ This is the "slope" vector function derived when the
  differential equation for the unconstrained physical
  pendulum is decoupled. }

  BEGIN
    uprime[1] := upass[2];
    uprime[2] := - pendconst * sin(upass[1])
  END;

{************************************************************}

PROCEDURE Airprime (pendconst, fairlength, airconst : real;
                    VAR upass, uprime : vector);

{ This is the "slope" vector function derived when the
  differential equation for the physical pendulum including
  air resistance is decoupled. }

  BEGIN
    uprime[1] := upass[2];
    IF upass[2] < 0
      THEN
        uprime[2] := -pendconst * sin(upass[1]) +
                     airconst * sqr(fairlength * upass[2])
      ELSE
        uprime[2] := -pendconst * sin(upass[1])
                   - airconst * sqr(fairlength * upass[2])
  END;

{************************************************************}
```

```
PROCEDURE Frprime (pendconst, frictconst : real; VAR upass,
                   uprime : vector);

{ This is the "slope" vector function derived when the
  differential equation for the physical pendulum including
  friction is decoupled. }

  BEGIN
    uprime[1] := upass[2];
    IF upass[2] < 0
      THEN
        uprime[2]:= -pendconst * sin(upass[1]) + frictconst
      ELSE
        uprime[2]:= -pendconst * sin(upass[1]) - frictconst
  END;

{*********************************************************}

PROCEDURE Bothprime (pendconst, fairlength, frictconst,
                     airconst: real;VAR upass,uprime: vector);

{ This is the "slope" vector function derived when the
  differentail equation for the physical pendulum including
  friction and air resistance is decoupled}

  BEGIN
    uprime[1] := upass[2];
    IF upass[2] < 0
      THEN
        uprime[2] := -pendconst * sin(upass[1]) + airconst
                     * sqr(fairlength * upass[2]) + frictconst
      ELSE
        uprime[2] := -pendconst * sin(upass[1]) - airconst
                     * sqr(fairlength * upass[2]) - frictconst
  END;

{*********************************************************}

PROCEDURE Runge4 (pendconst,airlength,frictconst,airconst,
                  delta_t: real;select: char;VAR ypast:
                  pastpoints; i : integer);

{ This is the fourth order Runge-Kutta method used to start
  the fourth order predictor-corrector method used.  It
  returns up to three back values for the function and its
  first derivative  given the values from the point before
  in the array ypast.  }

  VAR
    ytemp, k1, k2, k3, k4 : vector;
    j : integer;
```

```
BEGIN
 CASE select OF
  'p','P','n','N' :
   BEGIN
    Noprime (pendconst, ypast[i], k1);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k1[j]/2;
    Noprime (pendconst, ytemp, k2);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k2[j]/2;
    Noprime (pendconst, ytemp, k3);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k3[j];
    Noprime (pendconst, ytemp, k4);
    FOR j := 1 TO 2 DO
     ypast[i+1,j] := ypast[i,j] + delta_t *(k1[j] +
                     2*k2[j] + 2*k3[j] +k4[j])/6;
   END; { case of P, N }
  'a','A' :
   BEGIN
    Airprime (pendconst,airlength,airconst,ypast[i],
            k1);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k1[j]/2;
    Airprime (pendconst,airlength,airconst,ytemp,k2);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k2[j]/2;
    Airprime (pendconst,airlength,airconst,ytemp,k3);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k3[j];
    Airprime (pendconst,airlength,airconst,ytemp,k4);
    FOR j := 1 TO 2 DO
     ypast[i+1,j] := ypast[i,j] + delta_t *(k1[j] +
                     2*k2[j] + 2*k3[j] +k4[j])/6;
   END; { case of A }
  'f','F' :
   BEGIN
    Frprime (pendconst, frictconst, ypast[i], k1);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k1[j]/2;
    Frprime (pendconst, frictconst, ytemp, k2);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k2[j]/2;
    Frprime (pendconst, frictconst, ytemp, k3);
    FOR j := 1 TO 2 DO
     ytemp[j] := ypast[i,j] + delta_t * k3[j];
    Frprime (pendconst, frictconst, ytemp, k4);
    FOR j := 1 TO 2 DO
     ypast[i+1,j] := ypast[i,j] + delta_t *(k1[j] +
                     2*k2[j] + 2*k3[j] +k4[j])/6;
   END; { case of F }
  'b','B' :
```

```
      BEGIN
        Bothprime (pendconst, airlength, frictconst,
                airconst, ypast[i], k1);
        FOR j := 1 TO 2 DO
          ytemp[j] := ypast[i,j] + delta_t * k1[j]/2;
        Bothprime (pendconst, airlength, frictconst,
                airconst, ytemp, k2);
        FOR j := 1 TO 2 DO
          ytemp[j] := ypast[i,j] + delta_t * k2[j]/2;
        Bothprime (pendconst, airlength, frictconst,
                airconst, ytemp, k3);
        FOR j := 1 TO 2 DO
          ytemp[j] := ypast[i,j] + delta_t * k3[j];
        Bothprime (pendconst, airlength, frictconst,
                airconst, ytemp, k4);
        FOR j := 1 TO 2 DO
          ypast[i+1,j] := ypast[i,j] + delta_t *(k1[j] +
                          2*k2[j] + 2*k3[j] +k4[j])/6;
      END; { case of B }
    END; { case }
  END; { Runge4 }

{*********************************************************}

PROCEDURE PredictCorrect (pendconst, airlength, frictconst,
                    airconst, delta_t: real; select :
                    char; VAR ypast : pastpoints; VAR
                    k1, k2, k3 : vector);

  VAR
    k4, ytemp : vector;
    j : integer;

  BEGIN
    FOR j := 1 TO 2 DO    { predictor }
      ytemp[j]:= ypast[3,j] + delta_t *(23*k3[j] - 16*k2[j]
              + 5*k1[j])/12;
    CASE select OF
      'p','P','n','N' :  Noprime (pendconst, ytemp, k4);
      'a','A' :  Airprime (pendconst, airlength, airconst,
                      ytemp, k4);
      'f','F' :  Frprime (pendconst,frictconst, ytemp, k4);
      'b','B' :  Bothprime (pendconst,airlength,frictconst,
                       airconst,ytemp,k4);
    END; { case }
    FOR j := 1 TO 2 DO    { corrector - single iteration }
      ypast[4,j]:= ypast[3,j] + delta_t *(5*k4[j] + 8*k3[j]
                - k2[j])/12;
    k1 := k2; { Move functions up for next pass }
    k2 := k3;
    k3 := k4;
    ypast[3] := ypast[4];  { Move to next point }
```

```
  END; { PredictCorrect }

{*************************************************************}

PROCEDURE Multistep (pendconst,airlength,theta,frictconst,
                     airconst,delta_t: real; curve,select:
                     char; VAR simdata : datafile);

{ This is the numerical calculation for the simulation
  using a fourth order predictor-corrector method
  constructed from a three stage Adams-Bashforth method and
  a two stage Adams-Moulton method. The resulting multistage
  method is not self starting and requires three back
  points in order to begin predicting and correcting. The
  three back points are provided by the initial conditions
  and the fourth order Runge-Kutta method employed above.
  The error from the Runge-Kutta method is as good as the
  predictor-corrector method being used so no problems
  should arise as a result of the generation of the first
  three points. It is relatively fast (5 seconds for 600
  data points) and has fourth order accuracy. }


  VAR
    k1, k2, k3, k4, ytemp, buffer : vector;
    ypast : pastpoints;
    time : real;
    i, j, subint, int : integer;

  BEGIN
    rewrite (simdata);
    IF delta_t < 0.01 THEN
      BEGIN
        subint := ROUND(0.01/delta_t);
        int := 640;
      END
    ELSE
      BEGIN
        subint := 1;
        int := ROUND(6.4/delta_t);
      END;
    buffer[1] := ord(curve);
    buffer[2] := 0.0;
    write (simdata, buffer);
    buffer[1] := ord(select); { label data }
    buffer[2] := pendconst;
    ytemp[1] := airlength;
    ytemp[2] := theta;
    write (simdata, buffer, ytemp);
    ytemp[1] := frictconst;
    ytemp[2] := airconst;
```

```
  write (simdata, ytemp);
  time := 0.0;
  ypast[1,1] := theta;
  ypast[1,2] := 0.0;
  buffer[1] := time;
  IF curve IN ['d','D']
    THEN buffer[2] := ypast[1,1]
    ELSE buffer[2] := 0.0;
  write (simdata, buffer);
{ writeln (LST,buffer[1],'      ',buffer[2]);}
  FOR i := 1 TO 2 DO{ Calculate first three back points.}
    BEGIN
      time := time + delta_t;
      Runge4 (pendconst, airlength, frictconst, airconst,
            delta_t, select, ypast, i);
      IF delta_t >= 0.01 THEN
        BEGIN
          buffer[1] := time;
          IF curve IN ['d','D']
            THEN buffer[2] := ypast[i+1,1]
            ELSE buffer[2] := ypast[i+1,2];
          write (simdata, buffer);
        { writeln (LST,buffer[1],'      ',buffer[2]);}
        END; { if }
    END; { for i }
  CASE select OF
    'p','P','n','N' :
      BEGIN
        Noprime(pendconst,ypast[1],k1);{ The initial }
        Noprime(pendconst,ypast[2],k2);{ functional }
        Noprime(pendconst,ypast[3],k3);{ values for use }
      END; { case of P }              { in PredictCorrect }
    'a','A' :
      BEGIN
        Airprime (pendconst,airlength,airconst,ypast[1],
                k1);
        Airprime (pendconst,airlength,airconst,ypast[2],
                k2);
        Airprime (pendconst,airlength,airconst,ypast[3],
                k3);
      END; { case of A }
    'f','F' :
      BEGIN
        Frprime (pendconst, frictconst, ypast[1], k1);
        Frprime (pendconst, frictconst, ypast[2], k2);
        Frprime (pendconst, frictconst, ypast[3], k3);
      END; { case of F }
    'b','B' :
      BEGIN
        Bothprime (pendconst, airlength, frictconst,
                airconst, ypast[1], k1);
        Bothprime (pendconst, airlength, frictconst,
```

```
                airconst, ypast[2], k2);
        Bothprime (pendconst, airlength, frictconst,
                airconst, ypast[3], k3);
      END; { case of B }
    END; { case }
  FOR i := 1 TO int DO
    BEGIN
      FOR j := 1 TO subint DO
        BEGIN
          time := time + delta_t;
          PredictCorrect (pendconst,airlength,frictconst,
                        airconst,delta_t,select, ypast,
                        k1, k2, k3);
        END; { for j }
      buffer[1] := time;
      IF curve IN ['d','D']
        THEN buffer[2] := ypast[4,1]
        ELSE buffer[2] := ypast[4,2];
      write (simdata, buffer);
    { writeln (LST,buffer[1],'     ',buffer[2]);}
    END; { for i }
  close (simdata);
END; { Multistep }

{*********************************************************}

PROCEDURE Simulate (VAR simcomplete : boolean; VAR simdata:
                datafile);

{ This procedure runs the simulation when the user requests
  the simulation option from the main menu. }

  LABEL 2;

  CONST
    g = 9.81;

  VAR
    curve, select : char;
    imoment, mass, airlength, theta, cmdistance,
    delta_t : real;
    pconst, frictconst, airconst : real;

  BEGIN
    writeln ('The simulation will proceed based on the');
    writeln ('physcical configuration you are using.');
2 : writeln;
    writeln ('Enter the system configuration:');
    writeln ('     P - Simple Pendulum');
    writeln ('     N - No constraints (friction or air)');
    writeln ('     A - Air resistance only');
    writeln ('     F - Friction only');
```

```
writeln ('     B - Both air resistance and friction');
readln (select);
CASE select OF
 'p','P' :
  BEGIN
    writeln('Enter the length of the pendulum in');
    writeln('meters.');
    readln (cmdistance);
    writeln ('Enter the initial angle in radians.');
    readln (theta);
    writeln ('Enter the step size, delta t, to be ');
    writeln ('used in the simulation.(Multiples of');
    writeln ('.01 or divisors of .01 down to .001');
    writeln ('only for graphing purposes.)');
    readln (delta_t);
    writeln ('Do you wish to plot the displacement');
    writeln ('or velocity. Enter:');
    writeln ('     D - Displacement');
    writeln ('     V - Velocity');
    readln (curve);
    pconst := g/cmdistance;
    Multistep (pconst, 0, theta, 0, 0, delta_t,curve,
            select, simdata);
  END; { case of P }
 'n','N','a','A','f','F','b','B' :
  BEGIN
    airlength := 0.0;
    airconst := 0.0;
    frictconst := 0.0;
    writeln('Enter the pendulum mass (in kg).');
    readln (mass);
    writeln ('Enter the distance from the pivot to');
    writeln ('the center of mass.');
    readln (cmdistance);
    writeln ('Enter the moment of inertia for the');
    writeln ('pendulum.');
    readln (imoment);
    IF select IN ['a','A'] THEN
      BEGIN
        writeln ('Enter the distance to the center');
        writeln ('of the square plates.');
        readln (airlength);
        writeln ('Enter the coefficient of air');
        writeln ('resistance.');
        readln (airconst);
      END;
    IF select IN ['f','F'] THEN
      BEGIN
        writeln ('Enter the friction coefficient.');
        readln (frictconst);
      END;
    IF select IN ['b','B'] THEN
```

```
          BEGIN
            writeln ('Enter the distance to the center');
            writeln ('of the square plates.');
            readln (airlength);
            writeln ('Enter the coefficient of air');
            writeln ('resistance.');
            readln (airconst);
            writeln ('Enter the friction coefficient.');
            readln (frictconst)
          END;
        writeln ('Enter the initial angle in radians.');
        readln (theta);
        pconst := mass * g * cmdistance / imoment;
        writeln ('Enter the step size, delta t, to be');
        writeln ('used in the simulation.Multiples of');
        writeln ('.01 or divisors of .01 down to .001');
        writeln ('for graphing purposes.)');
        readln (delta_t);
        writeln ('Do you wish to plot the displacement');
        writeln ('or velocity. Enter:');
        writeln ('      D - Displacement');
        writeln ('      V - Velocity');
        readln (curve);
        Multistep (pconst,airlength,theta,frictconst,
                airconst,delta_t,curve,select,simdata)
      END { case of NOT P }
    ELSE
      BEGIN
        writeln ('Not a valid selection.');
        GOTO 2
      END; { else }
  END; { case }
  simcomplete := true;
END; { Simulate }

{************************************************************}

PROCEDURE Compare (VAR pendata : datafile);

{ This procedure compares the measured data from the lab
  with the motion calculated by the numerical method. It
  calculates the moment of inertia, coefficient of air
  resistance or the coefficient of friction for the
  pendulum by fitting the simulated curve to the curve of
  the actual motion.  This routine is the slowest because
  the simulation must be recalculated iteratively as the
  value of imoment, aircon or frictconst is a variable and
  must be incremented until a proper fit is obtained. }

  LABEL 3, 4, 5, 6;

  CONST
```

```
    g = 9.81;
    xtolerance = 0.004;
    delta_t = 0.01;

  VAR
    cselect, calcvar : char;
    intolerance : boolean;
    mass, imoment, airlength, theta, cmdistance, pendconst,
    frictconst, airconst, ytolerance, ilimit,flimit,alimit,
    iincrement,fincrement,aincrement,istart,fstart, astart,
    time : real;
    buffer, k1, k2, k3 : vector;
    ypast : pastpoints;
    i : integer;

  BEGIN
3 : writeln ('In what physical configuration was the ');
    writeln ('data taken?');
    writeln ('     N - No constraints');
    writeln ('     A - Air resistance only');
    writeln ('     F - Friction effects only');
    writeln ('     B - Both friction and air resistance');
    writeln ('     Q - Quit and return to the main menu.');
    readln (cselect);
    IF cselect IN ['n','N','a','A','f','F','b','B']
      THEN
        BEGIN
          reset (pendata);
          read (pendata, buffer);{skip velocity plot label}
          read (pendata, buffer);
          IF round(buffer[1])  ord(cselect) {see if the }
            THEN        { data was taken in the indicated }
              BEGIN                        { configuration. }
                writeln ('The data was not taken in the "',
                        cselect,'" configuration.');
                GOTO 3
              END; { then }
      END { then }
    ELSE IF cselect IN ['q','Q']
      THEN GOTO 5
      ELSE
        BEGIN
          writeln ('Response not allowed.');
          GOTO 3;
        END; { else }
  6: writeln ('Which constant do you wish to calculate?');
    writeln ('Enter:');
    writeln ('     I - Moment of inertia');
    writeln ('     F - Coefficient of friction');
    writeln ('     A - Coefficient of air resistance');
    readln (calcvar);
    IF NOT (calcvar IN ['i','I','f','F','a','A']) THEN
```

```
  BEGIN
    writeln ('Not an allowed response.');
    GOTO 6
  END; { IF NOT }
writeln('The comparison procedure requires certain');
writeln ('incremental values, starting points and');
writeln ('limits. The first value needed is the ');
writeln ('tolerance for the accuracy of the');
writeln('calculated omega, or y coordinate. Enter');
writeln ('the y tolerance (.2 is a good start).');
readln (ytolerance);
writeln ('The next entries deal with the starting');
writeln('value or lower limit, an increment and an');
writeln ('upper limit to the value of the constant');
writeln ('to be calculated.');
CASE calcvar OF
  'a','A' :
    BEGIN
      writeln ('Enter the starting value for the');
      writeln ('coefficient of air resistance.');
      readln (astart);
      writeln ('Enter the amount to be incremented');
      writeln ('each pass.  WARNING: The smaller');
      writeln ('the increment the longer the');
      writeln ('calculation.');
      readln (aincrement);
      writeln ('Enter the upper limit on the');
      writeln ('coefficient of air');
      writeln ('resistance.');
      readln (alimit);
    END; { case of A }
  'f','F' :
    BEGIN
      writeln ('Enter the starting value for the');
      writeln ('coefficient of');
      writeln ('friction.');
      readln (fstart);
      writeln ('Enter the amount to be incremented');
      writeln ('each pass.  WARNING: The smaller');
      writeln ('the increment the longer the');
      writeln ('calculation.');
      readln (fincrement);
      writeln ('Enter the upper limit for the');
      writeln ('coefficient of friction.');
      readln (flimit);
    END; { case of F }
  'i','I' :
    BEGIN
      writeln ('Enter the starting value for the');
      writeln ('moment of inertia.');
      readln (istart);
      writeln ('Enter the amount to be incremented');
```

```
      writeln ('each pass.  WARNING: The smaller');
      writeln ('the increment the longer the');
      writeln ('calculation.');
      readln (iincrement);
      writeln ('Enter the upper limit for the');
      writeln ('moment of inertia.');
      readln (ilimit);
    END; { case of I }
  END; { case }
writeln ('Enter the total mass of the pendulum.');
readln (mass);
writeln ('Enter the distance from the pivot to the');
writeln ('center of mass.');
readln (cmdistance);
imoment := istart; airconst := astart;
frictconst := fstart;
CASE cselect OF
  'a','A' :
    BEGIN
      CASE calcvar OF
        'a','A' :
          BEGIN
            writeln ('Enter the distance to the');
            writeln ('center of the plates.');
            readln (airlength);
            writeln ('Enter the moment of inertia');
            writeln ('for the pendulum.');
            readln (imoment)
          END; { case of A }
        'f','F' :
          BEGIN
            writeln('Friction cannot be considered');
            writeln ('in this set up.');
            GOTO 3
          END; { case of F }
        'i','I' :
          BEGIN
            writeln ('Enter the distance to the');
            writeln ('center of the plates.');
            readln (airlength);
            writeln ('Enter the coefficient of air');
            writeln ('resistance.');
            readln (airconst);
          END  { case of I }
      END; { case }
    END; { case of A }
  'f','F' :
    BEGIN
      CASE calcvar OF
        'a','A' :
          BEGIN
            writeln ('Air resistance is not');
```

```
          writeln ('considered in this set up.');
          GOTO 3
        END; { case of A }
      'f','F' :
        BEGIN
          writeln ('Enter the moment of inertia');
          writeln ('for the pendulum.');
          readln (imoment)
        END; { case of F }
      'i','I' :
        BEGIN
          writeln ('Enter the coefficient of');
          writeln ('friction.');
          readln (frictconst)
        END; { case of I }
    END; { case }
  END; { case of F }
'n','N' :
  BEGIN
    IF calcvar IN ['a','A','f','F'] THEN
      BEGIN
        writeln ('Neither air resistance or');
        writeln ('friction are considered in ');
        writeln ('this configuration.');
        GOTO 3
      END; { then }
  END; { case of N }
'b','B' :
  BEGIN
    writeln ('Enter the distance to the center ');
    writeln ('of the plates.');
    readln (airlength);
    CASE calcvar OF
      'a','A' :
        BEGIN
          writeln ('Enter the coefficient of');
          writeln ('friction.');
          readln (frictconst);
          writeln ('Enter the moment of inertia');
          writeln ('for the pendulum.');
          readln (imoment)
        END; { case of A }
      'f','F' :
        BEGIN
          writeln ('Enter the coefficient of air');
          writeln ('resistance.');
          readln (airconst);
          writeln ('Enter the moment of inertia');
          writeln ('for the pendulum.');
          readln (imoment)
        END; { case of F }
      'i','I' :
```

```
            BEGIN
              writeln ('Enter the coefficient of');
              writeln ('friction.');
              readln (frictconst);
              writeln ('Enter the coefficient of air');
              writeln ('resistance.');
              readln (airconst)
            END; { case of I }
        END; { case }
      END; { case of B }
    END; { case }
    writeln ('Enter the initial angle in radians.');
    readln (theta);
    pendconst := mass * g * cmdistance / imoment;
4 : CASE calcvar OF
      'a','A' :
        BEGIN
          airconst := airconst + aincrement;
          IF airconst > alimit THEN
            BEGIN
              writeln ('The coefficient of air');
              writeln ('resistance exceeds the limit.');
              writeln ('Either the tolerance is too');
              writeln ('strict or the value is not ');
              writeln ('between the start value and ');
              writeln ('the upper limit.');
              GOTO 5
            END;
        END; { case of A }
      'f','F' :
        BEGIN
          frictconst := frictconst + fincrement;
          IF frictconst > flimit THEN
            BEGIN
              writeln ('The coefficient of friction');
              writeln ('exceeds the limit.  Either the');
              writeln ('tolerance is too strict or the');
              writeln ('actual value is not between ');
              writeln ('the start value and the limit.');
              GOTO 5
            END;
        END; { case of F }
      'i','I' :
        BEGIN
          imoment := imoment + iincrement;
          pendconst := mass * g * cmdistance / imoment;
          IF imoment > ilimit THEN
            BEGIN
              writeln ('The moment of inertia exceeds');
              writeln('the limit. Either the tolerance');
              writeln ('is too strict or the actual');
              writeln ('value is not between the start');
```

```
        writeln ('value and the upper limit.');
        GOTO 5
      END;
    END; { case of I }
END; { case }
seek (pendata, 6); { Skip labels and the first few }
ypast[1,1] := theta; { points calculated by the }
ypast[1,2] := 0.0;  { Runge-Kutta method which are }
time := 0.0;        { difficult to incorporate into }
FOR i := 1 TO 2 DO { the comparison. }
 BEGIN
  time := time + delta_t;
  Runge4 (pendconst,airlength,frictconst,airconst,
        delta_t,cselect,ypast,i);
 END; { for i }
CASE cselect OF
 'p','P','n','N' :
  BEGIN
   Noprime (pendconst, ypast[1], k1);
   Noprime (pendconst, ypast[2], k2);
   Noprime (pendconst, ypast[3], k3);
  END; { case of P }
 'a','A' :
  BEGIN
   Airprime (pendconst,airlength,airconst,
          ypast[1], k1);
   Airprime (pendconst,airlength,airconst,
          ypast[2], k2);
   Airprime (pendconst,airlength,airconst,
          ypast[3], k3);
  END; { case of A }
 'f','F' :
  BEGIN
   Frprime (pendconst, frictconst, ypast[1], k1);
   Frprime (pendconst, frictconst, ypast[2], k2);
   Frprime (pendconst, frictconst, ypast[3], k3);
  END; { case of F }
 'b','B' :
  BEGIN
   Bothprime (pendconst, airlength, frictconst,
          airconst, ypast[1],k1);
   Bothprime (pendconst, airlength, frictconst,
          airconst, ypast[2],k2);
   Bothprime (pendconst, airlength, frictconst,
          airconst, ypast[3],k3);
  END; { case of B }
 END; { case }
WHILE NOT eof(pendata) DO
 BEGIN
  read (pendata, buffer);
  intolerance := false;
  WHILE NOT intolerance DO
```

```
          BEGIN
            time := time + delta_t;
            PredictCorrect(pendconst,airlength,
                           frictconst,airconst,delta_t,
                           cselect,ypast,k1, k2, k3);
            IF abs(buffer[1] - time) < xtolerance
              THEN intolerance := true;
          END; { while not }
        IF abs(buffer[2] - ypast[4,2]) >= ytolerance
          THEN GOTO 4
      END; { while not }
    close (pendata);
    CASE calcvar OF
      'a','A' :
        BEGIN
          writeln ('The coefficient of air resistance');
          writeln ('for this system is:');
          writeln (airconst:6:4)
        END;
      'f','F' :
        BEGIN
          writeln ('The coefficient of friction for');
          writeln ('this system is:');
          writeln (frictconst:6:4)
        END;
      'i','I' :
        BEGIN
          writeln ('The moment of inertia for this');
          writeln ('system is:');
          writeln (imoment:6:4,' kg-m**2')
        END;
    END; { case }
  5:  writeln;
  END; { Compare }

{************************************************************}

PROCEDURE Compplot (VAR file1, file2 : datafile);

{ This routine plots two data files for comparison.  It
  uses the organic Turbo Pascal routines discussed in the
  comments for the Plot Procedure. }

  LABEL 1;

  VAR
    x1, x2, y1, y2, config, scale : integer;
    buff, temp, temp2 : vector;
    curve : char;

  BEGIN
    clrscr;
```

```
graphmode;
hires;
Axis;
reset (file1);
read (file1,temp);
curve := chr(round(temp[1]));
IF curve IN ['d','D']
  THEN scale := 40
  ELSE scale := 10;
read (file1, temp, buff, temp2);  { read labels }
config := round(temp[1]);
writeln (curve,' PLOT 1: Config = ',chr(config),
        ' m*g*d/I = ',temp[2]:5:3,' Airlength = ',
        buff[1]:5:3,' Theta(0) = ', buff[2]:5:3);
writeln ('        Frict Coef = ',temp2[1]:5:3,
        ' Air Coef = ',temp2[2]:5:3);
read (file1,buff);
x2 := round(buff[1] * 100) + 10;
y2 := 110 - round(buff[2] * scale);
WHILE NOT eof(file1) DO
  BEGIN
    x1 := x2;
    y1 := y2;
    read (file1, buff);
    x2 := round(buff[1] * 100) + 10;
    y2 := 110 - round(buff[2] * scale);
    draw (x1, y1, x2, y2, 5);
  END; { while not }
close (file1);
reset (file2);
read (file2, temp);
curve := chr(round(temp[1]));
IF curve IN ['d','D']
  THEN scale := 40
  ELSE scale := 10;
read (file2, temp, buff, temp2);
config := round(temp[1]);
writeln (curve,' PLOT 2: Config = ',chr(config),
        ' m*g*d/I = ',temp[2]:5:3,' Airlength = ',
        buff[1]:5:3,' Theta(0) = ', buff[2]:5:3);
writeln ('        Frict Coef = ',temp2[1]:5:3,
        ' Air Coef = ',temp2[2]:5:3);
read (file2,buff);
x2 := round(buff[1] * 100) + 10;
y2 := 110 - round(buff[2] * scale);
WHILE NOT eof(file2) DO
  BEGIN
    x1 := x2;
    y1 := y2;
    read (file2, buff);
    x2 := round(buff[1] * 100) + 10;
    y2 := 110 - round(buff[2] * scale);
```

```
      draw (x1, y1, x2, y2, 5);
    END; { while not }
  close (file2);
1 : writeln ('Are you done with the plot? (Y/N)');
  IF Yes
    THEN textmode
    ELSE GOTO 1
END; { Compplot }

{*********************************************************}

PROCEDURE Compdata (VAR file1, file2 : datafile);

{ This routine plots the data and simulation files for
  comparison.  File1 is the data file.  It uses the
  organic Turbo Pascal routines discussed in the comments
  for the Plot Procedure. }

  LABEL 1;

  VAR
    x1, x2, y1, y2, config, scale : integer;
    buff, temp, temp2 : vector;
    curve : char;

  BEGIN
    clrscr;
    graphmode;
    hires;
    Axis;
    reset (file1);
    read (file1,temp);
    curve := chr(round(temp[1]));
    read (file1, temp, buff, temp2);
    config := round(temp[1]); { Get labeling data }
    writeln('Data Config = ',chr(config),' m*g*d/I = ',
          temp[2]:5:3,'Airlength = ',buff[1]:5:3,
          ' Theta(0) = ',buff[2]:5:3);
    WHILE NOT eof(file1) DO
      BEGIN
        read (file1, buff);
        x2 := round(buff[1] * 100) + 10;
        y2 := 110 - round(buff[2] * 10);
        draw (x2, y2-2, x2, y2+2, 5);
        draw (x2-4, y2, x2+4, y2, 5);
      END; { while }
    close (file1);
    reset (file2);
    read (file2, temp);
    curve := chr(round(temp[1]));
    IF curve IN ['d','D']
      THEN scale := 40
```

```
    ELSE scale := 10;
  read (file2, temp, buff, temp2);
  config := round(temp[1]);
  writeln (curve,' Plot Config = ',chr(config),
        ' m*g*d/I = ',temp[2]:5:3,' Airlength = ',
        buff[1]:5:3,' Theta(0) = ', buff[2]:5:3);
  writeln ('       Frict Coef = ',temp2[1]:5:3,
        ' Air Coef = ',temp2[2]:5:3);
  read (file2,buff);
  x2 := round(buff[1] * 100) + 10;
  y2 := 110 - round(buff[2] * scale);
  WHILE NOT eof(file2) DO
    BEGIN
      x1 := x2;
      y1 := y2;
      read (file2, buff);
      x2 := round(buff[1] * 100) + 10;
      y2 := 110 - round(buff[2] * scale);
      draw (x1, y1, x2, y2, 5);
    END; { while not }
  close (file2);
1 : writeln ('Are you done with the plot? (Y/N)');
  IF Yes
    THEN textmode
    ELSE GOTO 1
END; { Compdata }

{************************************************************}

BEGIN  { Physpendulum }
  assign (simdata1, 'A:SIMDATA1.DAT');{ This is Turbo }
  assign (simdata2, 'A:SIMDATA2.DAT');{ Pascal's method }
  assign (pendata, 'A:PENDATA.DAT'); {of assigning file }
  datataken := false; { variables to disk files, not by }
  sim1complete := false;  { rewrites. }
  sim2complete := false;
  writeln ('Welcome to the Physical Pendulum data ');
  writeln ('collection and simulation program.');
1 : writeln;
  writeln ('Enter T to Takedata, S to Simulate, C to');
  writeln ('Compare the data with the numerical');
  writeln ('solution and calculate the moment of ');
  writeln ('inertia, A to calculate data using the');
  writeln ('Small Angle approximation, or Q to Quit.');
  readln (response);
  CASE response OF
    'q','Q' : HALT;
    'a','A' :
      BEGIN
        writeln('The small angle approximation routine');
        writeln('has been calculated assuming a system');
        writeln('of no constraints (no air resistance ');
```

```
    writeln ('or friction).  If comparisons');
    writeln('evaluating the appropriate use of the');
    writeln ('small angle approximation are going ');
    writeln ('to be made, insure that the');
    writeln ('simulations are done using the No');
    writeln ('constraints (N) option.  The small');
    writeln ('angle routine stores its data in the');
    writeln ('first simulation file so comparison ');
    writeln ('simulations are second simulations.');
    Smallangle (simdata1);
    sim1complete := true;
    writeln ('Do you wish to plot the results');
    writeln ('obtained using the small angle');
    writeln ('approximation? (Y/N)');
    IF Yes
      THEN Plot (simdata1);
  END; { case of A }
't','T' :
  BEGIN
    IF datataken
      THEN
        BEGIN
          writeln('Any new data taken will destroy');
          writeln ('the current data.  Do you wish');
          writeln ('to take new data? (Y/N)');
          IF Yes
            THEN Takedata (datataken)
            ELSE GOTO 1
        END  { then }
      ELSE Takedata (datataken);
    writeln ('Do you wish to plot the data? (Y/N)');
    IF Yes
      THEN DataPlot (pendata);
    writeln ('Do you wish to shift the data? (Y/N)');
    IF Yes THEN
      BEGIN
    3 : Shiftdata (pendata);
        writeln ('Do you wish to plot the shifted');
        writeln ('data? (Y/N)');
        IF Yes
          THEN DataPlot (pendata);
        writeln ('Do you wish to shift the data');
        writeln ('again? (Y/N)');
        IF Yes
          THEN GOTO 3;
      END;
  END; { case of T }
's','S' :
  BEGIN
    IF sim1complete
      THEN
        BEGIN
```

```
        writeln ('A simulation has been ');
    2 : writeln ('completed.  Do you wish to');
        writeln ('resimulate, calculate a second');
        writeln ('simulation or return to the');
        writeln ('main menu?  Enter');
        writeln (' R - Resimulate ');
        writeln (' S - Second simulation ');
        writeln (' Q - Quit, return to main menu');
        readln (simselect);
        CASE simselect OF
          'q','Q' : GOTO 1;
          'r','R' :
            BEGIN
              Simulate (sim1complete, simdata1);
              writeln ('Do you wish to plot the');
              writeln ('simulation? (Y/N)');
              IF Yes
                THEN Plot (simdata1);
            END; { case of R }
          's','S' :
            BEGIN
              Simulate (sim2complete, simdata2);
              writeln ('Do you wish to plot the');
              writeln ('simulation? (Y/N)');
              IF Yes
                THEN Plot (simdata2);
            END  { case of S }
          ELSE
            BEGIN
              writeln ('Not an allowed response.');
              GOTO 2
            END; { else }
        END; { case }
      END { then }
    ELSE
      BEGIN
        Simulate (sim1complete,simdata1);
        writeln ('Do you wish to plot the');
        writeln ('simulation?(Y/N)');
        IF Yes
          THEN Plot (simdata1);
      END; { else }
  END; { case of S }
'c','C' :
  BEGIN
    IF datataken
      THEN
        BEGIN
          Compare (pendata);
          GOTO 1
        END { then }
      ELSE
```

```
        BEGIN
          writeln ('Data must first be taken in');
          writeln ('order to run the comparison');
          writeln ('and calculate any of the');
          writeln ('constants.');
          GOTO 1
        END; { else }
    END { case of C }
  ELSE
    BEGIN
      writeln ('Not a proper response.');
      GOTO 1
    END; { else }
END; { case }
IF datataken AND sim1complete
  THEN
    BEGIN
      writeln ('Do you wish to graphically compare');
      writeln ('the plot of the data and the first');
      writeln ('simulation? (Y/N)');
      IF Yes THEN
        Compdata (pendata, simdata1);
    END; { then }
IF datataken AND sim2complete
  THEN
    BEGIN
      writeln ('Do you wish to graphically compare');
      writeln ('the plot of the data and the second');
      writeln ('simulation? (Y/N)');
      IF Yes THEN
        Compdata (pendata, simdata2);
    END; { then }
IF sim1complete AND sim2complete
  THEN
    BEGIN
      writeln('Do you wish to graphically compare');
      writeln('the two simulations? (Y/N)');
      IF Yes
        THEN Compplot (simdata1, simdata2)
    END; { then }
  GOTO 1
END. { Physpendulum }
```

```
*
* SUBROUTINES FOR "DROP"
* MODIFIED FOR USE IN "FRESHPEN"
*
* 10/19/85
*
* BRIAN DAVIS
* MODIFIED BY MARK STEVENS
* 8/19/87
*
*
* PUBLIC ROUTINES
*
 PUBLIC SETUP
 PUBLIC DRAWCURS  DRAW CURSOR
 PUBLIC COUNT  COLLECT DATA
 POUBLIC GRAPH  GRAPH DATA
*
*EXTERNAL REFERENCES
*
PUTC EXT
SETT EXT
*
*
* TIME LOCATION : CHANGE TO LOW MEMORY IF ROM'ED
*
TIME RMB 1
*
SETUP ORCC #$50
 LEAX IRQ,PCR
 STX $10D
 LDA #$7E
 STA $10C
 LDA #$BF
 STA $FF52 PB6= INPUT
 CLR $FF50
 RTS
*
* INTERRUPT HANDLER
*
IRQ TST $FF02 CLEAR INTERRUPT FLAG
 DEC TIME,PCR
 RTI
*
*
* PASCAL CALL:
* PROCEDURE COUNT (STARTADD:INTEGER)
*
*
* STARTADD IS STARTING ADDRESS OF BUFFER
* BUFFER : ARRAY OF INTEGER
```

```
*
* INITIAL STACK:
STARTADD EQU 6 START OF BUFFER ADDRESS
*
* 4-5,S: LINK
* 2-3,S: RETURN ADDRESS
* 0-1,S: SAVED U
*
* SETUP
*
COUNT PSHS U SAVE LINK
 LDU STARTADD,S
 LDX #$FF01 TURN OFF PIA INTERRUPTS
 BSR CLRINT
 INC $FF03 60 HZ IRQ ON
 ANDCC #$EF ENABLE INTERRUPTS
 LDA #64
 PSHS A COUNT ON STACK
 LDX #$FF50
 SYNC WAIT UNTIL IRQ
TRAN CLRB TRANSITIONS = 0
 LDA #$6
 STA TIME,PCR COUNT FOR 1/10 SECOND
 LDA ,X GET PORT VALUE
TRAN1 CMPA ,X CHANGE?
 BEQ TRAN2
 INCB YES, INCREMENT TRANSITION COUNT
 LDA ,X NEW PORT VALUE
TRAN2 TST TIME,PCR CHECK TIME
 BNE TRAN1
 CLR ,U+ CLEAR HIGH BYTE OF INTEGER
 STB ,U+ STORE POINT
 DEC ,S SEE IF DONE
 BNE TRAN
 ORCC #$50
 PULS A,U,PC
*
*
*
* CLRINT CLEARS INTERRUPTS
* X = ADDRESS OF PIA CRA
*
CLRINT LDA ,X
 ANDA #$FE
 STA ,X
 LDA 2,X
 ANDA #$FE
 STA 2,X
 RTS
*
* PASCAL CALL:
* PROCEDURE GRAPH (STARTADR : INTEGER)
```

```
* STARTADR IS AN ADDRESS : BUFFER[1]
*
* PLOTS LOW ORDER BYTES OF ARRAY
* PLOTS 64 POINTS, IE
* BUFFER[1] - BUFFER[64]
*
* STACK:
START EQU 6
*
* 4-5,S: LINK
* 2-3,S: RETURN ADDRESS
* 0-1,S: SAVED U
*
* $88 CONTAINS POINTER TO CURRENT
* GRAPHICS CONTROL BLOCK
*
* FORMAT OF GRAPHICS CONTROL BLOCK;
* (ADDRESS IN Y REG)
*
* 0-1, Y = SCREEN START ADDRESS
* 2-3, Y = SCREEN END ADDRESS
* 4, Y = GRAPHICS Y COORD.
* 5, Y = GRAPHICS COORD.
*
*
*
GRAPH PSHS U SAVE LINK
 LDU START,S U = START ADDRESS
 LDY $88 Y = CONTROL BLOCK POINTER
 CLR 5,Y X COORD = 0
*
* LOOP TO PLOT/UNPLOT
*
 LEAU 1,U POINT TO LOW BYTE
GLP LDA ,U++
 LDB #180 SCALE Y
 MUL
 SUBA #191
 COMA
 STA 4,Y STORE Y
 LBSR SETT GET THE ADDRESS 'BIT
 COMB FLIP BIT MASK
 ANDB ,X RESET BIT
 STB ,X PUT IT ON SCREEN
 LDA 5,Y X = X+4
 ADDA #4
 STA 5,Y
 BNE GLP
GRAPHRET PULS U,PC
*
*
* PASCAL CALL DRAWCURS(X,Y:INTEGER)
```

```
*
* DRAWS CURSOR AT COORDINATES
* PERFORMS SCALING
*
* 7,S:X COORD
* 5,S:Y COORD
*
DRAWCURS LDY $88
 LDA 5,S GET Y
 LDB#180 SCALE IT
 MUL
 SUBA #186 CURSOR IS 4 BYTES HIGH- 186 ADJUSTS
 COMA  FOR HEIGHT SO BOTTOM IS ABOVE POINT
 LDB 7,S X = X+4
 LSLB
 LSLB
 STD 4,Y PUT INTO GY,GX
 PSHS U SAVE LINK
 LBSR SETT
 LDA #4
 LEAU ARROW,PCR
 PSHS D
CURS1 LDA ,U+
 LDB 1,S
 MUL
 ASLB
 ROLA
 EORA ,X
 EORB 1,X
 STD ,X
 LEAX $20,X
 CMPX 2,Y
 BHI CURSEX
 DEC ,S
 BNE CURS1
CURSEX PULS D,U,PC
ARROW FDB $10A0
 FDB $C0E0
*
*
*
 END
```

APPENDIX C: Example Plots

Any plot the Physpend routine displays on the screen of the Zenith Z-140 PC can be printed using the print screen command and a graphics capable printer. Before running Physpend the DOS command "graphics" must be executed. The following example plots include a data plot, a comparison of data to a simulation, a comparison of two simulations, a plot of the displacement and angular velocity for the same system, and an example of solution instability with increasing step size.

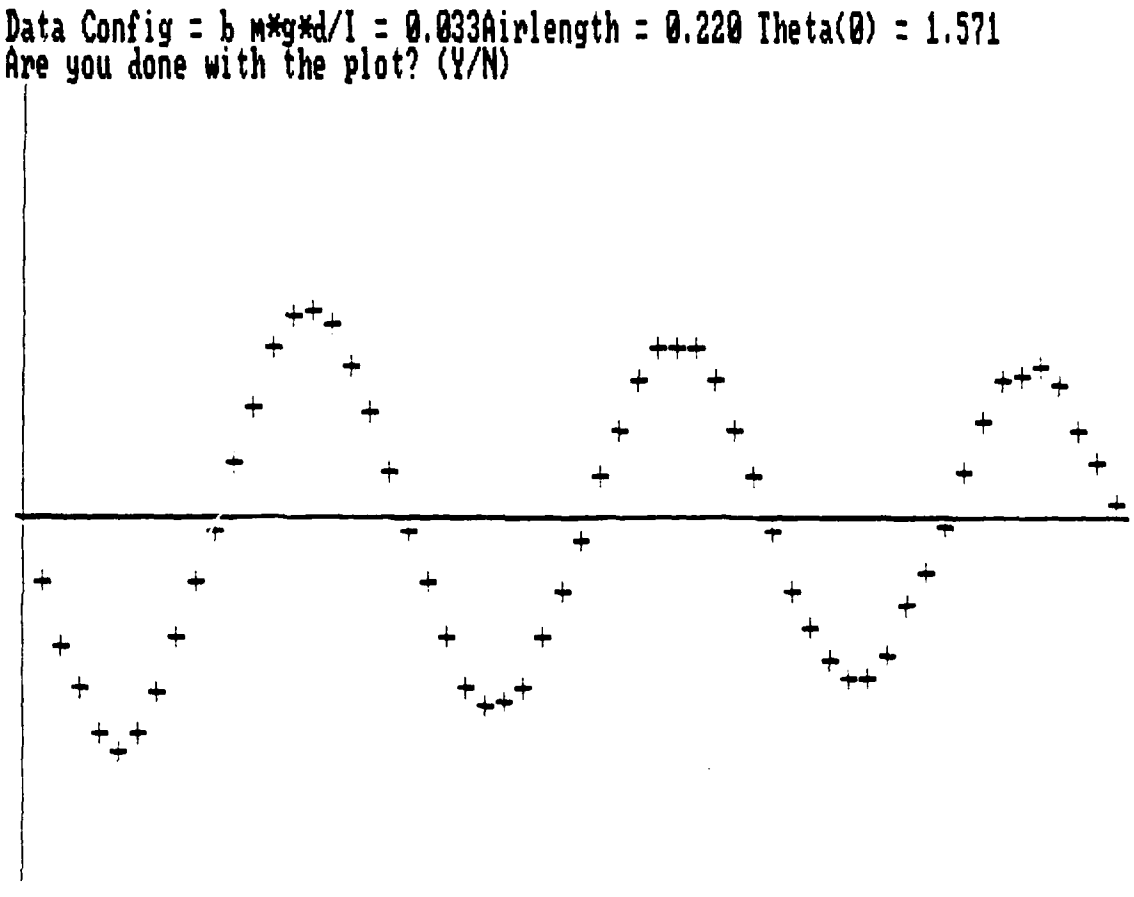


Figure C.1. Data Plot

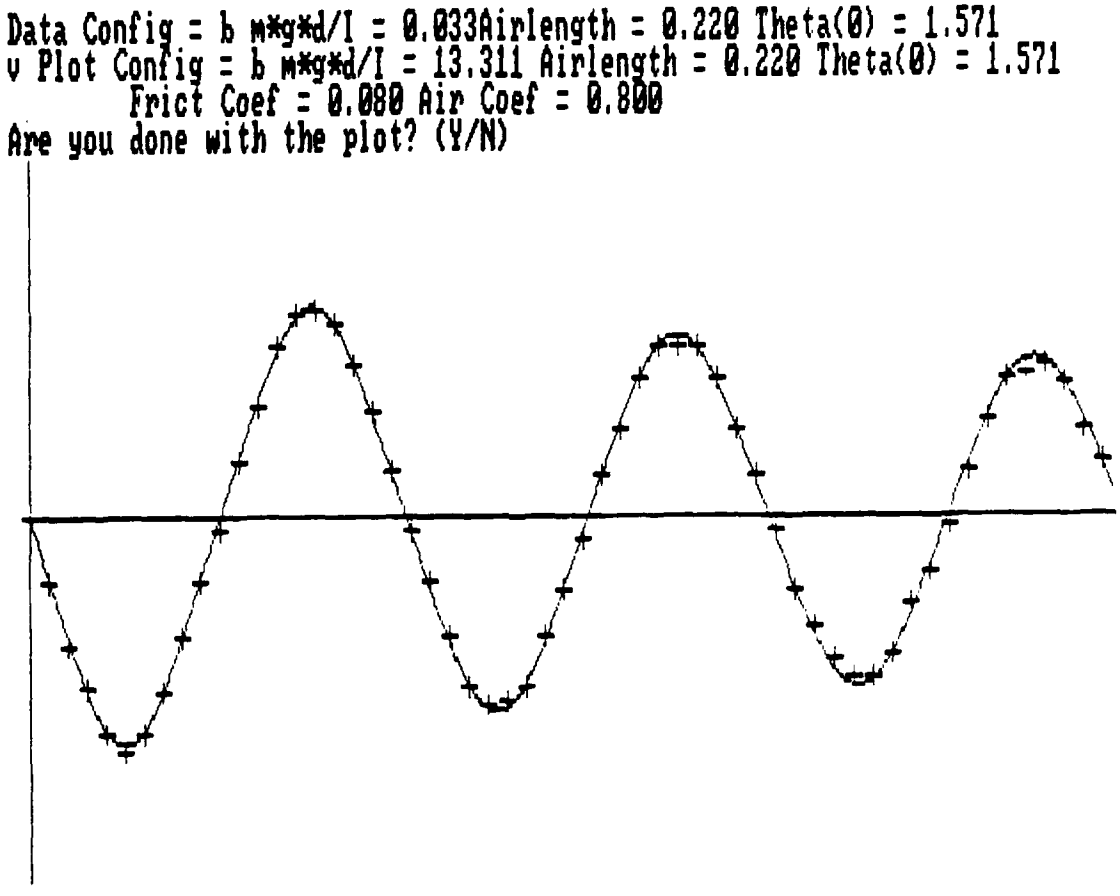


Figure C.2. Data vs. Simulation

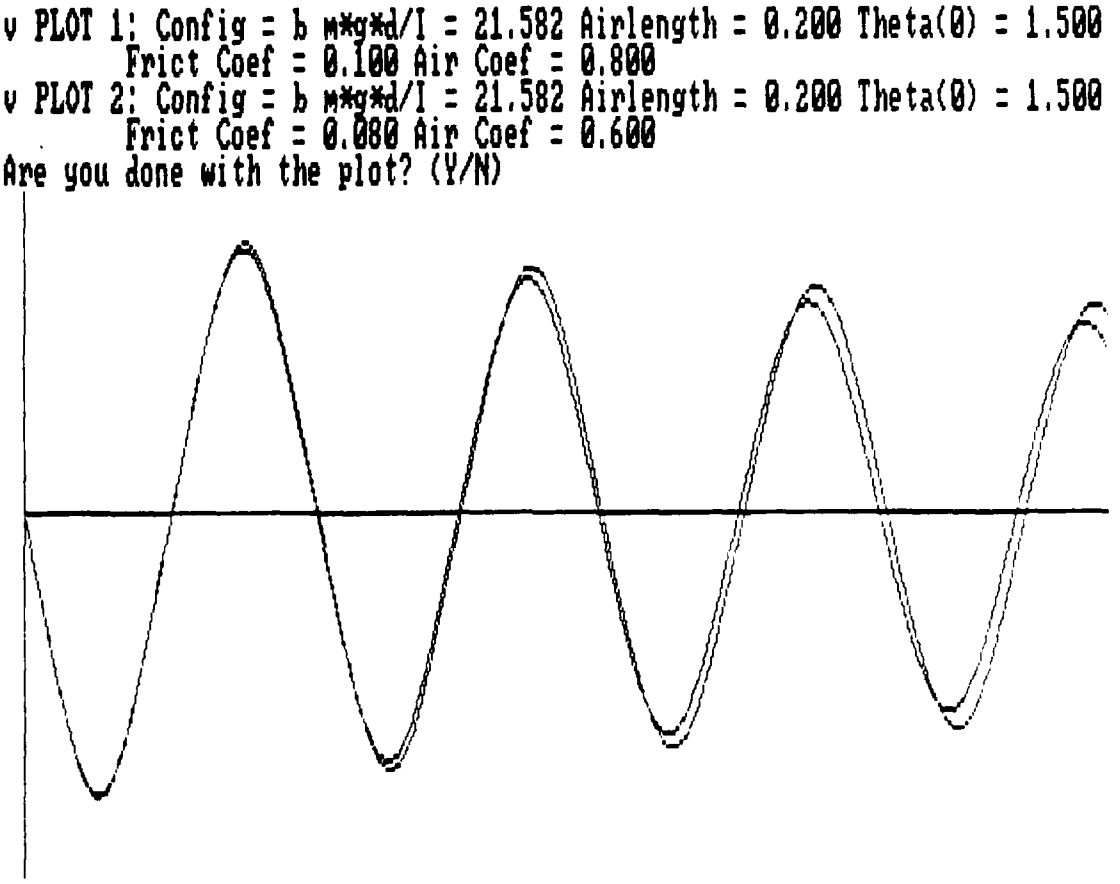


Figure C.3. Simulation 1 vs. Simulation 2

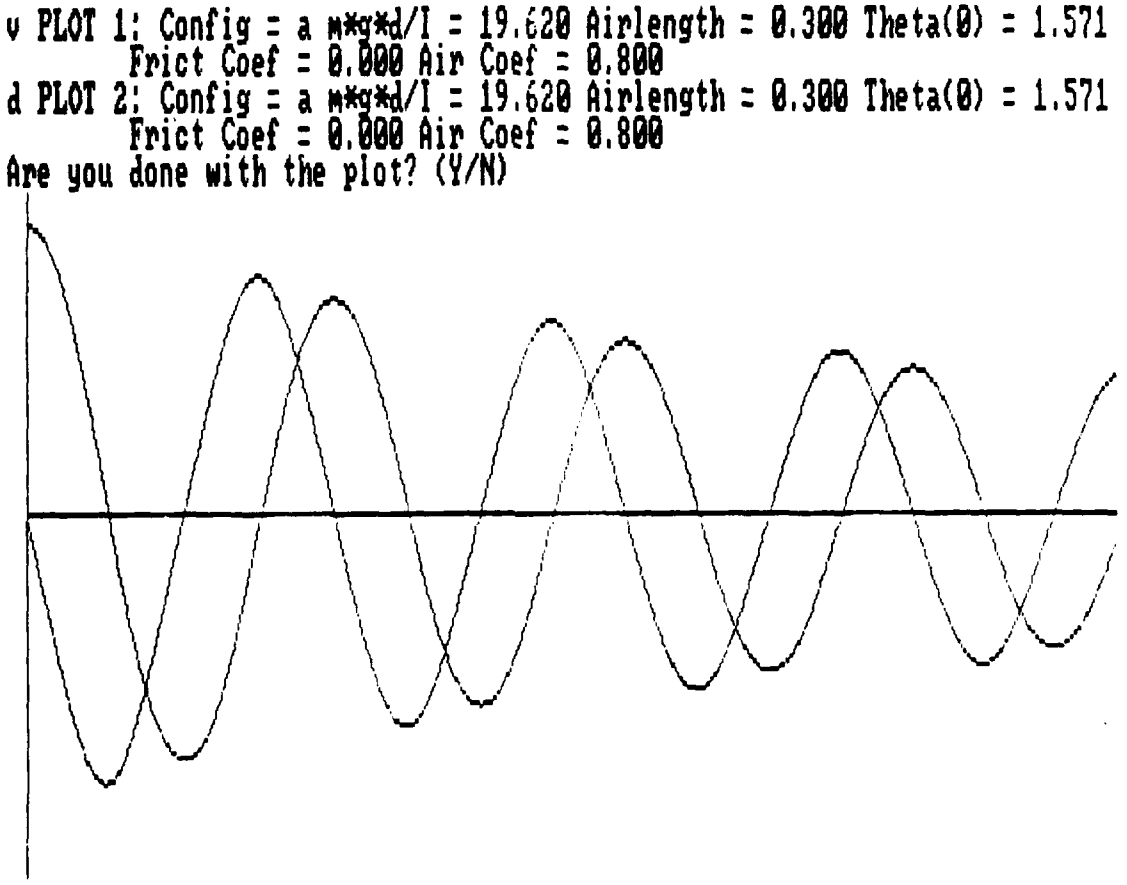


Figure C.4. Angular Velocity vs. Displacement

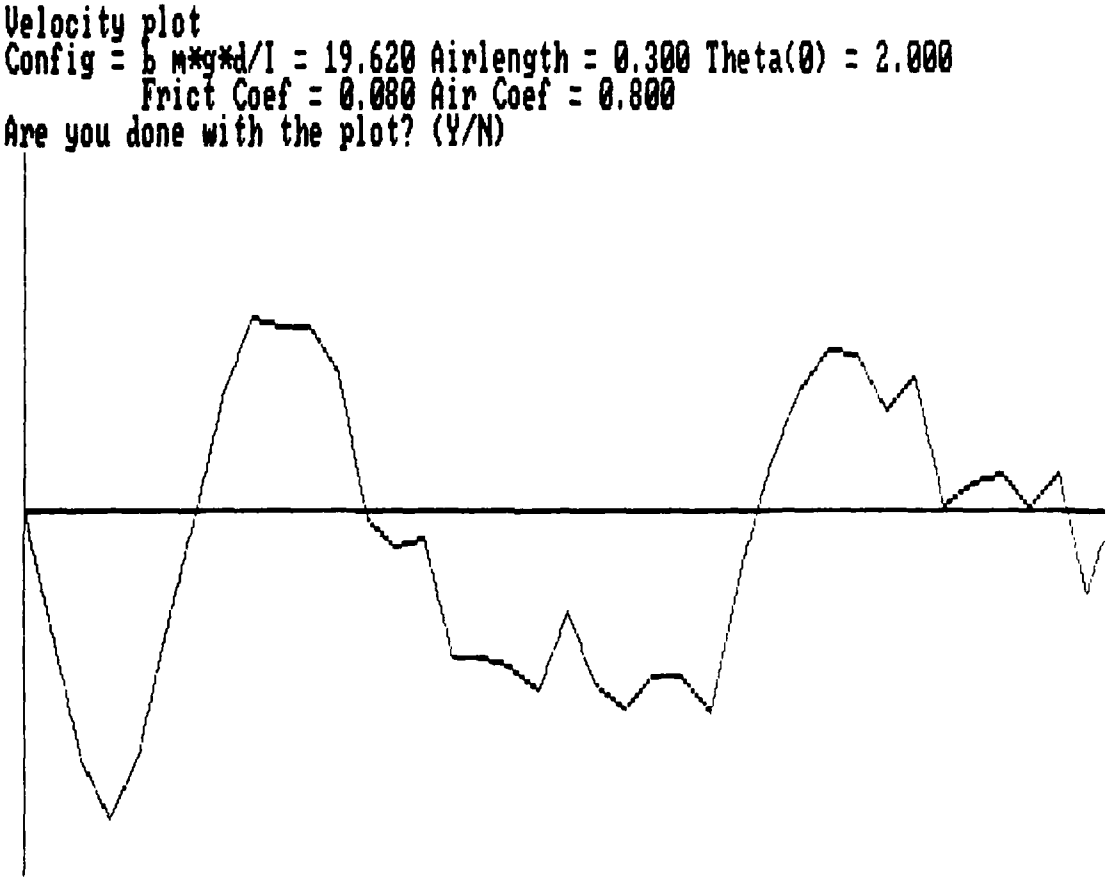


Figure C.5. Numerical Method Instability

APPENDIX D: Experiment Instructions

Experimental background and instructions were developed for each of the experiments. The instructions for all three experiments are included in this appendix. The level of difficulty and detail of the instructions were matched to the expected capabilities of the students. The freshmen are expected to complete the experiment and turn in their laboratory notebooks after two ninety minute periods, while the juniors are given one week and are required to submit a more formal report.

NUMERICAL METHODS AND ROTATIONAL MOTION
(VISCOUS DAMPING - TERMINAL VELOCITY)

I. INTRODUCTION

In this experiment we will consider the rotational motion of a disk with adjustable plates which resist the disk's motion through the air. The disk will be accelerated by applying a constant torque. The rotational inertia of the rigid body will be calculated by breaking it into parts and finding the rotational inertia of each by knowing the mass and shape of each. The total rotational inertia is simply the sum of the individual parts.

After calculating the moment of inertia the angular velocity plot of the system will be calculated using the improved Euler's method and a comparison made to the actual angular velocity curve measured by the microcomputer. By adjusting the fins the effect of air resistance on the velocity of the wheel will be investigated.

The apparatus consists of a thirty centimeter aluminum disk and a hub of smaller diameter, both mounted on an axle with ball bearings. We also have a notched disk which allows us to measure the rotation of the body and two adjustable rectangular plates provide air resistance.

II. THEORY REVIEW

The viscous damping force that affects the disk can be approximated as being proportional to some power of the linear velocity of the plates. The total torque acting on the disk can be written

$$I\frac{d\omega}{dt} = \frac{mgrI}{I+mr^2} - kl(\omega)^n$$

where k is a constant which describes the damping force, l is the distance from the rotational axis to the center of the plates, r is the moment arm of the accelerating weight, and n is an integer. As the disk begins to accelerate under an applied torque the viscous damping force applies a retarding torque slowing down the angular acceleration. Finally, the disk reaches a contant "terminal" angular velocity at which point the net torque acting on the disk is zero.

The exact solution of this motion in practice is quite difficult. The applicaton of the improved Euler's method to this motion lets us study the behavior of this system in detail. We can see the effect of varying k and n and compare the calculated solution to the actual motion as measured by the microcomputer.

III. PROCEDURE AND ANALYSIS

Part A

To find the rotational inertia of the system from the mass and shape of the individual parts use the unassembled parts found next to the balance. DO NOT DISASSEMBLE THE APPARATUS TO BE USED IN PART C. Measure the mass and

all relevant dimensions and compute the resulting rotatinal inertia. Estimate the uncertainty for each of the parts and the overall uncertainty of the calculated value of the rotational inertia. Show all of the measurements, calculations and uncertainties in the form of a table.

Part B

Noting in this case that

$$\frac{d\omega}{dt} = \frac{mgr}{I+mr^2} - kl(\omega)^n$$

we can apply the improved Euler's method to calculate and plot the angular velocity of the system. Remember that the improved Euler's method first computes the value of angular velocity one half of the way into the first interval.

$$\omega_{0.5} = \omega_0 + \alpha_0(\frac{\Delta t}{2})$$

Then using this value we calculate

$$\alpha_{0.5} = \frac{mgr}{I+mr^2} - kl(\omega_{0.5})^n$$

Finally we obtain

$$\omega_1 = \omega_0 + \alpha_{0.5}\Delta t$$

To calculate the value of ω for the next point the initial value ω_0 is replaced with ω_1 and the above steps are repeated. Notice that we are simply using the slope of the angular velocity curve one half way through the time interval and we add that change to the preceding value of ω at the beginning of the interval. Calculate and plot the first twenty points for the motion using t = 0.1 seconds and k = 0.4. Measure l and d from the apparatus and use your calculated value for the rotational inertia from Part A. You might want to tabulate your values for α and ω to aid in calculation and plotting.

Will increasing the length of the time interval increase or decrease the accuracy of the numerical method in approximating the actual motion? Why?

Part C

We are using the microcomputer to measure the angular velocity of the rotating disk. It will also use an equivalent numerical method to calculate the angular velocity curve for comparison to the data. The microcomputer will count slots in the notched wheel every 0.1 seconds. It plots the resulting angular velocity curve, the slope of which is the angular acceleration, α.

Turn on the microcomputer and initiate the Airwheel program. Follow the instructions given and enter the appropriate data when prompted.

First, orient the plates so that they cause a minimum amount of air resistance. Wind up the string with the 500 gram mass. Release the wheel and simultaneously press any key on the microcomputer to take the data. Input your values from Part A and Part B in the simulation, choosing the no air resistance option, and compare the calculated curve with the actual motion. Are the curves the same? If not, why?

Adjust the input value for I until you match the actual angular velocity curve. Record this value.

Part D

Take data again but adjust the plates for maximum air resistance. Run the simulation now using I from part C. Again try to match the calculated curve with the actual angular velocity curve by varying values of k and n. Which model for the viscous force is closest to the actual motion? What is the value for k?

Adjust the plates to some intermediate position and measure the angular velocity. Again match the curve with the computer calculated angular velocity. Is the same model for the viscous force still the most accurate? What happens to the value of k? What happens to the value for the terminal angular velocity?

If you changed the mass accelerating the wheel would it affect the terminal velocity? How?

Part E

Test the accuracy of the numerical method when you vary the time interval. What happens to the value for the terminal velocity for large time intervals? How large can Δt be before the difference between the curves becomes significant? For a more dramatic demonstration of the sensitivity of this type of numerical method to the step size load the Freshpen program for a physical pendulum and vary the step size for the harmonic motion calculation. Why would the improved Euler's method be more sensitive to Δt for the oscillating function obtained for harmonic motion than to the angular velocity curve for the rotating disk that approaches a terminal value?

NUMERICAL METHODS AND THE PHYSICAL PENDULUM
(FREE AND DAMPED HARMONIC MOTION)

I. INTRODUCTION

In this experiment we will use numerical methods to investigate the oscillation of a physical pendulum. We will examine both undamped and damped oscillations. The rotational inertia of the pendulum will be calculated by breaking it into parts and finding the rotational inertia of each part by knowing the mass and shape of each. The total rotational inertia is simply the sum of the individual parts.

After calculating the moment of inertia the motion of the pendulum will be recorded by the microcomputer. A numerical method equivalent to the improved Euler's method (called Heun's method) is then employed by the computer to simulate the motion.

The apparatus consists of a thirty centimeter diameter aluminum disk with a hub and slotted disk for measuring rotation. It also has two rectangular plates which provide viscous damping and a weight to offset the center of mass from the axis of rotation.

II. THEORY REVIEW

For an undamped physical pendulum the equation of motion is

$$I\frac{d\omega}{dt} = -mgd\sin\theta$$

where d is the distance from the rotation axis and θ is measured from the vertical. In the case of viscous damping the force providing the damping torque can be approximated as being proportional to some power of the linear velocity of the resisting plates. This torque always opposes the direction of motion and modifies the equation of motion to

$$I\frac{d\omega}{dt} = -mgd\sin\theta \pm kl(\omega)^n$$

where l is the distance from the rotation axis to the center of the plates, k is a constantwhich describes the damping force, and n is an integer.

The exact solution to either of these equations is difficult, except when the small angle approximation ($\sin\theta = \theta$) is made for the undamped case. The application of the numerical method to this system allows us to study the motion in detail. The effects of varying n and k can be investigated and values for the actual system determined.

III. PROCEDURE AND ANALYSIS

Part A

To find the rotational inertia from the mass and shape of the individual parts use the unassembled parts found next to the balance. DO NOT DISASSEMBLE THE APPARATUS TO BE USED IN PART B. Measure the mass and all relevant dimensions and compute the resulting rotational inertia. (Note: The parallel axis theorem might be useful here.) Estimate the uncertainty for each of the parts and the

overall uncertainty of the calculated value of the rotational inertia. Show all of the measurements, calculations and uncertainties in the form of a table.

Calculate the distance from the rotation axis to the center of mass and record the total mass of the pendulum.

Part B

We are using the microcomputer to measure the angular velocity of the pendulum. It will also use the numerical method to calculate the angular velocity given input data and will compare it to the actual data curve. The microcomputer will count slots in the notched wheel every 0.1 seconds and will plot the resulting angular velocity. Note that it only plots the absolute value of the angular velocity.

Turn on the microcomputer and initiate the Freshpen program. Follow the instructions given and enter the appropriate data when prompted.

First, orient the plates so that they cause a minimum amount of air resistance. Set the pendulum to the starting angle for the desired motion. Make sure you measure and record the starting angle for the oscillation. Release the pendulum and simultaneously press any key on the microcomputer to take the data. Input your values from Part A into the simulation, choosing the no air resistance option, and compare the calculated curve to the actual motion. Are they the same? If not, why? What is the calculated period of the motion?

Adjust the input values for I until you match the actual angular velocity curve. Record this value. What is the period of the actual motion?

Part C

Take the data again but adjust the plates for maximum air resistance. Run the simulation now using I from Part B. Again match the calculated curve with the actual angular velocities by adjusting the values of k and n. What value of n seems to be the closest to the actual measured motion? What is the value for k? What is the period of motion for this configuration? Compare it to the period in Part B. Explain any differences.

Adjust the plates to an intermediate position and again measure the angular velocity with the microcomputer. Are the values for n and k the same? If not, why?

Part D

Use the simulation routine and determine a value for k that results in a critically damped system. How might you provide such damping in the laboratory? Discuss what happens to the period, rotational inertia, d, and the value of k if a heavier weight were used to unbalance the disk. Would critical damping require a larger or smaller value for k in this case?

Part E

Test the accuracy of the numerical method when you vary the time interval. What happens as you increase the step size? At what value for Δt does the calculated curve seem to develop a significant error? Why does this error occur? If the angular velocity function were not oscillatory and approached some asymptotic value would the numerical method be as sensitive to the size of the step in time? For verification load the Airwheel program and investigate the effect of changing step size in the terminal

velocity simulation.

NUMERICAL METHODS AND THE PHYSICAL PENDULUM

This experiment allows you to investigate the harmonic motion of a real physical pendulum utilizing a microcomputer to take the data and compare it to the calculated motion derived using numerical methods. The setup consists of two systems; the modified TRS-80 system for taking data, and the Zenith Z-140 PC used for analysis. Each time data is taken it must be input into the Zenith for analysis.

The data is taken by means of a notched wheel and two photogates. The motion of whatever physical configuration you construct is recorded by counting the tripping of the photogates as the notches pass. There are 90 notches on the wheel and the count is recorded every 0.1 second. The resulting data yields an angular velocity plot. Using the known angular displacement of each notch the integer counts are converted to radians per second and displayed for transfer into the Zenith. To take data turn on the TRS-80 system. Insert the program disk and type RUN "DUMP". Respond "0" to the drive prompt and enter "JPENDATA" for the filename. The down loaded program will then instruct you on the procedure for taking and displaying the data.

Once data is taken you will want to analyze it with the Zenith. To do this insert the Zenith program diskette. Turn the system on. At the prompt enter GRAPHICS. This will allow you to print the plots from the screen by using the PrtSc key. Then enter PHYSPEND. The program will then give you instructions for analysis. The program will accept the input data, calculate the motion for small angle approximations, motion with friction and air resistance and even the motion of a simple pendulum. You are able to compare two simulations, or either simulation with the data. Any plot can be printed with the Print Screen command. The simulation utilizes a 4th order Predictor-Corrector method to approximate the solution to the differential equations of motion (you should know what the equations are). The accuracy of this method can be tested by varying the time step size. It is accurate enough to calculate the actual values of different factors for the physical pendulum by using an iterative process. It will calculate the motion and compare it to the data point by point until it gets a fit, subject to the limits you give it.

Utilizing the computer and software as a tool you should ivestigate the physical pendulum for large and small angle oscillations, large and small moments of inertia, large and negligible friction effects and the simple pendulum and small angle approximations verses the actual system. None of the physics theory is difficult and you should readily verify the effects of varying parameters (such as the moment of inertia, friction coefficient, etc.) on the motion of the pendulum. What you investigate is up to you, but areas of interest include, but are not limited to,

- at what angle is the small angle approximation no longer appropriate?

- the acutal moment of inertia of the pendulum.

- the coefficient of friction in the bearings.

- the effects and coefficient of air resistance.

- variations in the accuracy of the numerical method with changing time intervals.

ENDNOTES

1 Robert Resnick and David Halliday, Physics, Part I, 3rd ed. (New York: John Wiley and Sons, 1977), pg 313.

2 Ibid., p. 323.

3 Ibid., p. 245.

4 Lee W. Johnson and R. Dean Riess, Numerical Analysis (Reading, Massachusetts: Addison-Wesley Publishing Company, 1982), p. 366.

5 Ibid., p. 354.

6 Ibid., p. 362.

7 Ibid., p. 367.

8 Ibid., p. 369.

9 Ibid., pp. 373-393.

10 Ibid., p. 399.

11 Ibid., p. 402.

12 Ibid., p. 408.

13 Ibid., p. 394.

14 DEFT Pascal Workbench User's Guide (: DEFT Systems Inc., 1984).

15 Jeff Duntemann, Complete Turbo Pascal (Glenview, Ilinois: Scott, Foresman and Company, 1986).

16 Lance Leventhal, 6809 Assembly Language Programming (Berkeley: Osborne/McGraw-Hill, 1981).

17 Peter Grogono, Programming in Pascal, 2nd ed. (Reading, Massachusetts: Addison-Wesley Publishing Company, 1980).

REFERENCES

Color Computer Disk System Owners Manual and Programing Guide. : Tandy Corporation, 1981.

DEFT Pascal Workbench User's Guide. : DEFT Systems Inc., 1984.

Duntemann, Jeff. Complete Turbo Pascal. 2nd ed. Glenview, Illinois: Scott, Foresman and Company, 1986.

Getting Started with Extended Color Basic. : Tandy Corporation, 1984.

Goldstein, Herbert. Classical Mechanics. 2nd ed. Reading, Massachusetts: Addison-Wesley Publishing Company, 1980.

Grogono, Peter. Programming in Pascal. 2nd ed. Reading, Massachusetts: Addison-Wesley Publishing Company, 1985.

Johnson, Lee W. and Riess, R. Dean. Numerical Analysis. Reading, Massachusetts: Addison-Wesley Publishing Company, 1982.

Leventhal, Lance. 6809 Assembly Language Programming. Berkeley: Osborne/McGraw-Hill, 1981.

Meiners, Harry F.; Eppenstein, Walter; Oliva, Ralph A.; and Shannon, Thomas. Laboratory Physics. 2nd ed. New York: John Wiley and Sons, 1987.

Resnick, Robert and Halliday, David. Physics, Part I. 3rd ed. New York: John Wiley and Sons, 1977.

Staugaard, Andrew C. Jr. TRS-80 Color Computer Interfacing with Experiments. Indianapolis: Howard W. Sams and Company, 1983.

END

DATE

FILMED

9-88

DTIC